commodore COMPUTER

Commodore Business Machines Pty. Ltd.
3, Campbell Street
Artarmon, N.S.W. 2064
Telephone (02) 437 6296
Telex 71375 Comdre

COMMODORE 64 SOFTWARE DEVELOPMENT KIT

CONTENTS

commodore

Computer Systems Group
487 Devon Park Drive
Wayne, PA 19087
(215) 687-9750

MAX SOFTWARE DEVELOPMENT PROTOCOLS

CONFIDENTIAL

SOFTWARE DEVELOPMENT PROTOCOLS -- THE MAX MACHINE

Following is a list of "software protocols" -- standards you should use when programming cartridge-based software for the Commodore VIC 20. These protocols standardize copyright notice, use of function keys, keyboard controls and special options.

1. OPENING DISPLAY

The opening display should include the copyright notice (see Number 2) and a program title. If it is known in advance that a program will be sold under different names in different countries, the name of the program may be excluded, but copyright notice must appear. The program should AUTOSTART and launch IMMEDIATELY INTO THE PROGRAM, with the copyright notice shown in the upper lefthand corner briefly, then disappearing as the game begins.

2. COPYRIGHT NOTICE

The proper copyright notice, which must appear in the opening display of all programs, is:

(c) 1982 **Commodore Electronics Ltd.**

If program memory space is tight, the following alternatives may be used:

(c) 1982 **Commodore Elec. Ltd**
(c) 1982 **Commodore**

3. STARTING THE PROGRAM

All cartridge games should run automatically (AUTOSTART) when the computer is first powered up. The protocols for STARTING a cartridge program are:

FROM THE KEYBOARD...use the f1 function key.
FROM THE JOYSTICK...use the fire button.
FROM THE PADDLE....use the fire button (and check POT value).

Cartridges which use either joystick or keyboard should start from the fire button, and restart between rounds the same way. Keyboard-only programs should use the f1 key to start or restart. Some programs may require the use of the f1 key to start the program initially, with the fire button used to restart the game between rounds.

Paddles and Joysticks. If you're testing for a paddle to start a game, you should also test the POT VALUE to see if there is a paddle being used (not just the pushbutton), otherwise the program might think you're starting with a joystick, since the paddle pushbutton is the same as "joystick left". If the POT value is not 255 then you have a paddle present.

4. USE OF GAME CONTROLS/PRIORITIES

If your program/game uses a SINGLE JOYSTICK or includes a ONE PLAYER OPTION calling for a single joystick, the program should look to Game Control Port A first and use that port for 1 player programs.

MAX joystick A is on the righthand of the keyboard and should be used for single player programs.

On the Commodore-64 there are 2 joystick ports on the right side of the computer. Joystick A is the port closest to the back of the machine.

5. SELF-ADJUSTING SCREEN

The "self adjusting" feature of the MAX MACHINE should automatically center the picture on te screen before the program begins. This centering occurs automatically in the MAX.

6. GAMES/HIGH SCORE

All games should record the previous "high score" for that game and display it either between rounds, or in the corner of the screen while the game is being played.

7. GAME "REWARDS"

As a general standard, computer games should include several levels of difficulty (usually as the game progresses) and one or more "reward" rounds which may take the form of a musical or visual display or a "bonus" round with special game action or higher rounds for reaching a higher score, new graphic characters introduced at higher levels, etc. Many games have special "hidden" rewards at higher rounds which prompt the user to play the game over and over again to gain expertise to reach those rounds.

SOFTWARE DEVELOPMENT PROTOCOLS -- THE MAX MACHINE

Following is a list of "software protocols" -- standards you should use when programming cartridge-based software for the Commodore VIC 20. These protocols standardize copyright notice, use of function keys, keyboard controls and special options.

1. OPENING DISPLAY

The opening display should include the copyright notice (see Number 2) and a program title. If it is known in advance that a program will be sold under different names in different countries, the name of the program may be excluded, but copyright notice must appear. The program should AUTOSTART and launch IMMEDIATELY INTO THE PROGRAM, with the copyright notice shown in the upper lefthand corner briefly, then disappearing as the game begins.

2. COPYRIGHT NOTICE

The proper copyright notice, which must appear in the opening display of all programs, is:

(c) 1982 Commodore Electronics Ltd.

If program memory space is tight, the following alternatives may be used:

(c) 1982 Commodore Elec. Ltd
(c) 1982 Commodore

3. STARTING THE PROGRAM

All cartridge games should run automatically (AUTOSTART) when the computer is first powered up. The protocols for STARTING a cartridge program are:

FROM THE KEYBOARD...use the f1 function key.
FROM THE JOYSTICK...use the fire button.
FROM THE PADDLE.....use the fire button (and check POT value).

Cartridges which use either joystick or keyboard should start from the fire button, and restart between rounds the same way. Keyboard-only programs should use the f1 key to start or restart. Some programs may require the use of the f1 key to start the program initially, with the fire button used to restart the game between rounds.

Paddles and Joysticks...If you're testing for a paddle to start a game, you should also test the POT VALUE to see if there is a paddle being used (not just the pushbutton), otherwise the program might think you're starting with a joystick, since the paddle pushbutton is the same as "joystick left". If the POT value is not 255 then you have a paddle present.

4. USE OF GAME CONTROLS/PRIORITIES

If your program/game uses a SINGLE JOYSTICK or includes a ONE PLAYER OPTION calling for a single joystick, the program should look to Game Control Port A first and use that port for 1 player programs.

MAX joystick A is on the righthand of the keyboard and should be used for single player programs.

On the Commodore-64 there are 2 joystick ports on the right side of the computer. Joystick A is the port closest to the back of the machine.

5. SELF-ADJUSTING SCREEN

The "self adjusting" feature of the MAX MACHINE should automatically center the picture on te screen before the program begins. This centering occurs automatically in the MAX.

6. GAMES/HIGH SCORE

All games should record the previous "high score" for that game and display it either between rounds, or in the corner of the screen while the game is being played.

7. GAME "REWARDS"

As a general standard, computer games should include several levels of difficulty (usually as the game progresses) and one or more "reward" rounds which may take the form of a musical or visual display or a "bonus" round with special game action or higher rounds for reaching a higher score, new graphic characters introduced at higher levels, etc. Many games have special "hidden" rewards at higher rounds which prompt the user to play the game over and over again to gain expertise to reach those rounds.

SECTION VI

COMMODORE 64 ASSEMBLER SYSTEM

SAMPLE ASSEMBLER SESSION

```
1.  LOAD "BOOT ALL",8          ;load editor/loader/DOS wedge

2.  RUN                        ;

3.  GET "CBM64.SRC"            ;load in a source program

4.                             ;edit your source program

5.  PUT "CBM64.SRC"            ;save your changes

6.  LOAD "ASM.C64",8,1         ;load Commodore 64 assembler

7.  RUN                        ;
```

Note: if you wish to assemble your source code without generating an object file, do not enter a file name when the assembler prompts you for object file name.

COMMODORE 64 ASSEMBLER SYSTEM

The COMMODORE 64 assembler system is an update of the CBM 8032 versison*. With the macro assembler and associated support programs, the user can now develop 6510 machine machine code directly on the COMMODORE 64. The assembler requires the 1541 disk drive and optionally a 1525 printer.

Included on the COMMODORE 64 developer's disk are the following programs:

DOS BOOT	boots the DOS wedge (RUN)
BOOT ALL	boots the editor/loader/DOS wedge (RUN)
ASM.C64	macro assembler (RUN)
LOLOAD.C64	loader at $0800 (RUN)
HILOAD.C64	loader at $C800 (SYS 51200)
EDIT.C64	editor for assembly source (SYS 49152)
XREF.C64	cross reference for assembler (RUN)
DOS 5.1	DOS manager (SYS 52224)
XVM4.8	40 column monitor at $8000 (SYS 32768)
XVM4.C	40 column monitor at $C000 (SYS 49152)
CBM64DOS.SRC	source code to DOS manager 5.1
CBM64 TO PET	short form PET emulator
BYEBYE	screen blank and restore routine

The following list shows the differences between old versions of the assembler system and the version for the COMMODORE 64.

Assembler

1. Centronics print option has been removed.
2. The screen is not cleared when the assembler is run.
3. Cannot run xref and object at the same time due to restrictions in the 1541.
4. This assembler has macros.
5. The screen is blanked when printing.

Loaders

1. Loaders print start address and end address of the load. It also prints "." on each record.
2. Loaders reside at $0801 and $C800.
3. Editor and loader can be co-resident.

Editor

1. The cold and the break commands have been removed.
2. This version does not scroll. (soon!)

* note: a copy of the CBM 8032 Assembler Development Package User's Manual can be obtained by contacting Stephen Murri at (215) 687-9880.

Table of Contents

INTRODUCTION

This manual describes the Assembly Language and assembly process for programs for the MCS-650X series of microprocessors. Several assemblers are available for program development, and while they are all slightly different in detail of use, they are the same in substance.

The process of translating a mnemonic or symbolic form of a computer program to actual machine code is called assembly, and a program which performs the translation is an assembler. The symbols used and rules of association for those symbols are the Assembly Language. In general, one Assembly Language statement will translate into one machine instruction. This distinguishes an assembler from a compiler which may produce many machine instructions from a single statement. An assembler which executes on a computer other than the one for which code is generated is called a cross-assembler. Use of cross-assemblers for program development for microprocessors is common since often a microcomputer system has fewer resources than are needed for an assembler. However, in the case of the Commodore system this is not true. With a floppy disk and printer the system is well suited for software development.

Normally digital computers use the binary number system for representation of data and instructions. Computers understand only ones and zeroes corresponding to an "on" or "off" state. Users, on the other hand, find it difficult to work with the binary number system and hence use a more convenient representation such as octal (base 8), decimal (base 10) or hexadecimal (base 16). Two representations of the MCS-650X operation to "load" information into an "accumulator" are:

```
10101001 (binary)
A9       (hexadecimal)
```

An instruction to move the value 21 (decimal) to the accumulator is:

```
A9 15 (hexadecimal)
```

Users still find numeric representations of instructions tedious to work with, and hence, have developed symbolic representations. For example, the preceding instruction might be written as:

```
LDA    #21
```

In this case, LDA is the symbol for A9, Load the Accumulator. A computer program used to translate the symbolic form LDA to numeric form A9 is called an assembler. The symbolic program is referred to as source code and the numeric program is the object code. Only object code can be executed on the processor.

Each machine instruction to be executed has a symbolic name referred to as an operation code (OPCODE). The OPCODE for "store accumulator" is STA. The OPCODE for "transfer accumulator to index X" is TAX. The 56 OPCODES for MCS-650X processors are detailed in Chapter 2. A machine instruction in

Assembly Language consists of an OPCODE and perhaps OPERANDS, which specify the data on which the operation is to be performed.

Instructions may be labelled for reference by other instructions as shown in:

L2 LDA #12

The label is L2, the OPCODE is LDA, and the OPERAND is #12. At least one blank must separate the three parts (fields) of the instruction. Additional blanks are inserted by the assembler for ease of reading. Instructions for the MCS-650X processors have at most one OPERAND and many have none. In these cases the operation to be performed is totally specified by the OPCODE as in CLC (Clear the Carry Bit).

Programming in Assembly Language requires learning the instruction set (OPCODES), addressing conventions for referencing data, the data structures within the processor, as well as the structure of Assembly Language programs. The user will be aided in this by reading and studying the MCS-650X hardware and programming manuals supplied with this development package.

INSTRUCTION FORMAT

Assembler instructions for the MCS-650X assembler are of two basic types according to function:

- Machine instructions, and
- Assembler directives

Machine instructions correspond to the 56 operations implemented on the MCS-650X processors. The instruction format is:

(label) (OPCODE) (OPERAND) (comments)

Fields are bracketed to indicate that they are optional. Labels and comments are always optional and many OPCODES such as RTS (Return from Subroutine) do not require OPERANDS. A typical instruction showing all four fields is:

```
LOOP   LDA   BETA,X   FETCH BETA INDEXED BY X
```

A field is defined as a string of characters separated by a blank space or tab characters. The list of OPCODES for the MCS-650X processors is shown in Table 1.

A label is an alphanumeric string of from one to six characters, the first of which must be alpha. A label may not be any of the 56 OPCODES, nor any of the special single characters, i.e. A, S, P, X or Y. These special characters are used by the assembler to reference the:

- Accumulator (A)
- Stack pointer (S)
- Processor status (P)
- Index registers (X and Y)

A label may begin in any column provided it is the first field of an instruction. Labels are used on instructions as branch targets and on data elements for reference in OPERANDS.

Table 1. MCS-650X Microprocessor Instruction Set - OPCODES

ADC	Add with Carry to Accumulator
AND	"AND" to Accumulator
ASL	Shift Left One Bit (Memory or Accumulator)
BCC	Branch On Carry Clear
BCS	Branch on Carry Set
BEQ	Branch on Zero Result
BIT	Test Bits in Memory with Accumulator
BMI	Branch on Result Minus
BNE	Branch on Result not Zero
BPL	Branch on Result Plus
BRK	Force an Interrupt or Break
BVC	Branch on Overflow Clear
BVS	Branch on Overflow Set
CLC	Clear Carry Flag
CLD	Clear Decimal Mode
CLI	Clear Interrupt Disable Bit
CLV	Clear Overflow Flag
CMP	Compare Memory and Accumulator
CPX	Compare Memory and Index X
CPY	Compare Memory and Index Y
DEC	Decrement Memory by One
DEX	Decrement Index X by One
DEY	Decrement Index Y by One
EOR	Exclusive-or Memory with Accumulator
INC	Increment Memory by One
INX	Increment X by One
INY	Increment Y by One
JMP	Jump to new Location
TXS	Transfer Index X to Stack Register
JSR	Jump to New Location Saving Return Address
LDA	Transfer Memory to Accumulator
LDX	Transfer Memory to Index X
LDY	Transfer Memory to Index Y
LSR	Shift One Bit Right (Memory or Accumulator)
NOP	Do Nothing - No Operation
ORA	"OR" Memory with Accumulator
PHA	Push Accumulator on Stack
PHP	Push Processor Status on Stack
PLA	Pull Accumulator from Stack
PLP	Pull Processor Status from Stack
ROL	Rotate One Bit Left (Memory or Accumulator)
ROR	Rotate One Bit Right (Memory or Accumulator)
RTI	Return From Interrupt
RTS	Return From Subroutine
SBC	Subtract Memory and Carry from Accumulator
SEC	Set Carry Flag
SED	Set Decimal Mode
SEI	Set Interrupt Disable Status
STA	Store Accumulator in Memory
STX	Store Index X in Memory
STY	Store Index Y in Memory
TAX	Transfer Accumulator to Index X
TAY	Transfer Accumulator to Index Y
TSX	Transfer Stack to Index X
TXA	Transfer Index X to Accumulator
TYA	Transfer Index Y to Accumulator

The OPERAND portion of an instruction specifies either an address or a value. An address may be computed by expression evaluation and the assembler allows considerable flexibility in expression formation. An Assembly Language expression consists of a string of names and constants separated by operators +, -, *, and / (add, subtract, multiply, and divide). Expressions are evaluated by the assembler to computer OPERAND addresses. Expressions are evaluated left to right with no operator precedence and no parenthetical grouping. Note that expressions are evaluated at assembly time and not execution time.

Any string of characters following the OPERAND field is considered a comment and is listed, but not further processed. If the first non-blank character of any record is a semi-colon (;), the record is processed as a comment. On the instructions which require no OPERAND, comments may follow the OPCODE. At lease one separating character (space or horizontal tab) must separate the fields of an instruction.

There are twelve assembler directives used to reserve storage and direct information to the assembler. Eleven have symbolic names with a period as the first character. The twelth, a symbolic equate, uses an equals sign (=) to establish a value for a symbol. A list of the directives are given below and their use is explained in a later section.

.IFE	.BYTE	.WORD	.DBYTE	.PAGE	.SKIP
.IFN	.OPT	.END	.FILE	.LIB	= or *=

Labels and symbols other than directives may not begin with a period.

A typical MCS-650X assembler program segment is shown in Table 2. This table is presented primarily to show the form of the information output by the assembler.

Table 2

SEGMENT OF AN MCS-650X PROGRAM

```
PET LOADER......PAGE 0002

LINE# LOC CODE LINE

0038 0293                             ;LINE OF BASIC TEXT TO ALLOW
0039 0293                             ;USER TO TYPE 'RUN'
0040 0293                                   * =$400
0041 0400 00                                 .BYT  0,13,4,10,0,158
;10 SYS
0041 0401 0D
0041 0402 04
0041 0403 0A
0041 0404 00
0041 0405 9E
0042 0406 28 31                             .BYT '(1039)',0,0,0
0042 040C 00
0042 040D 00
0042 040E 00


0044 040F A9 00                       LOAD  LDA #0
0045 0411 8D 81 02                          STA CHAN
0046 0414 8D 92 02                          STA OBJLEN
0047 0417 8D 7D 02                          STA RECCNT
0048 041A 8D 7E 02                          STA RECCNT+1
0049 041D A2 40                             LDX #OBJMSG-MSGS
0050 041F 20 4C 05                          JSR MSG
0051 0422 A2 28                             LDX #40
0052 0424 8E 7F 02                          STX INDEX
0053 0427 CE 7F 02                    LD10  DEC INDEX
0054 042A F0 E3                             BEQ LOAD
0055 042C 20 E1 F1                          JSR BASIN
0056 042F C9 20                             CMP #$20
0057 0431 F0 F4                             BEQ LD10
0058 0433 C9 0D                             CMP #$D
0059 0435 F0 D8                             BEQ LOAD
0060 0437 A2 00                             LDX #0
0061 0439 8E 92 02                          STX OBJLEN
0062 043C F0 07                             BEQ LD30
0063 043E 20 E1 F1                    LD20  JSR BASIN
0064 0441 C9 20                             CMP #$20
0065 0443 F0 15                             BEQ LD40
0066 0445 C9 0D                       LD30  CMP #$D
0067 0447 F0 11                             BEQ LD40
0068 0449 AE 92 02                          LDX OBJLEN
0069 044C E0 0E                             CPX #14
0070 044E F0 BF                             BEQ LOAD
0071 0450 9D 82 02                          STA OBJFIL,X
0072 0453 E8                                INX
0073 0454 8E 92 02                          STX OBJLEN
0074 0457 4C 3E 04                          JMP LD20
0075 045A A9 2C                       LD40  LDA #',
0076 045C 9D 82 02                          STA OBJFIL,X
0077 045F E8                                INX
0078 0460 A9 53                             LDA #'S
0079 0462 9D 82 02                          STA OBJFIL,X
0080 0465 E8                                INX
```

SYMBOLIC

Perhaps the most common OPERAND addressing mode is the symbolic form as in:

```
LDA   BETA    PUT BETA VALUE IN ACCUMULATOR
```

In the example BETA references a byte in memory that is to be loaded into the accumulator. BETA is an address at which the value is located. Similarly, in the instruction

```
LDA   ALPHA+BETA
```

the address ALPHA+BETA is computed by the assembler and the value at the computed address is loaded into the accumulator.

Memory associated with the MCS-650X processors is segmented into pages of 256 bytes each. The first page, page zero, is treated differently by the assembler and by the processor for optimization of memory storage space. Many of the instructions have alternate operation codes if the OPERAND address is in page zero memory. In those cases the address requires only one byte rather than the normal two. For example, if BETA is located at the byte 4B in page zero memory, then the code generated for

```
LDA   BETA
```

is A5 B4. This is called page zero addressing. If BETA is at 01 3C in memory page one the code generated is AD 3C 10. This is an example of absolute addressing. Thus, to optimize storage and execution time, a programmer should design with data areas in page zero memory whenever possible. Note that the assembler makes decisions on which form to use based on OPERAND address computation.

CONSTANTS

Constant values in Assembly Language can take several forms. If a constant is other than decimal a prefix character is used to specify type.

$	(Dollar sign) specifies hexadecimal
@	(Commercial at) specifies octal
%	(Percent) specifies binary
'	(Apostrophe) specifies an ASCII literal character in immediate instructions

The absence of a prefix symbol indicates decimal value. In the statement

```
LDA      BETA+5
```

the decimal number 5 is added to BETA to compute the address. Similarly;

```
LDA BETA+$5F
```

denotes that the hexadecimal value 5F is to be added to BETA for the address computation.

The immediate mode of addressing is signified by a # (a pounds

sign on some printers) followed by a constant.

Example: LDA #2

specifies that the decimal value 2 is to be put into the accumulator. Similarly;

LDA #'G

will load the ASCII character G into the accumulator.

Immediate mode addressing always generates two bytes of machine code, the OPCODE and the value to be used as OPERAND. Note that constant values can be used in address expressions and as values in immediate mode addressing. They can also be used to initialize locations as explained in a later section on assembler directives.

RELATIVE

There are 8 conditional branch instructions available to the user.

Example: BEQ START IF EQUAL BRANCH TO START

which might typically follow a compare instruction. If the values compared are equal a transfer to the instruction labelled START is made. The branch address is a one byte positive or negative offset which is added to the program counter during execution. At the time the addition is made the program counter is pointing to the next instruction beyond the branch instruction. Thus, a branch address must be within 127 bytes forward or 128 bytes backward from the conditional branch target instruction. An error will be flagged at assembly time if a branch target falls outside the bounds for relative addressing. Relative addressing is not used for any other instructions.

IMPLIED

Twenty-five instructions such as TAX (Transfer Accumulator to Index X) require no OPERAND and hence are single byte instructions. Thus, the OPERAND addresses are implied by the operation code.

Four instructions, ASL, LSR, ROL and ROR are special in that the accumulator, A, can be used as an OPERAND. In this special case, these four instructions are treated as implied mode addressing and only an operation code is generated.

INDEXED INDIRECT

In this mode the OPERAND address is a location in page zero memory which contains the address to be used as an OPERAND.

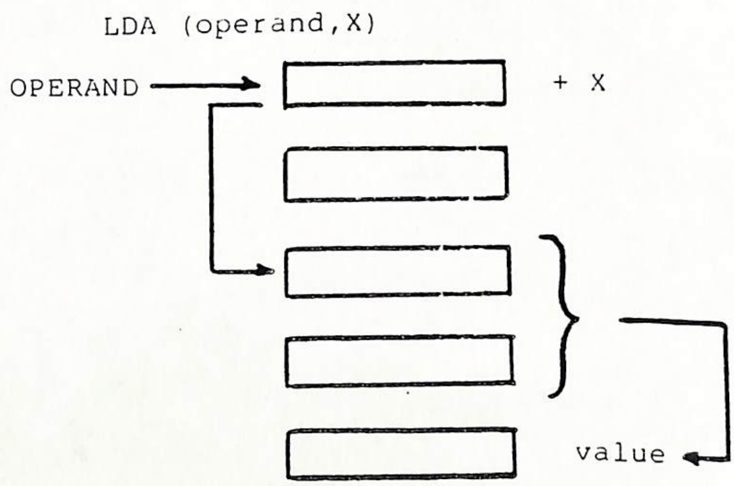
Example: LDA (BETA,X)

The parentheses around the OPERAND indicates indirect mode. In the above example the value in index register X is added to BETA. That sum must reference a location in page zero memory. During execution the high order byte of the address is ignored, thus forcing a page zero address. The two bytes starting at

that location in page zero memory are taken as the address of the OPERAND. For purposes of illustration assume the following:

BETA is 12
X contains 4
Locations 0017 and 0016 are 01 and 25
Location 0125 contains 37

Then BETA + X is 16, the address at location 16 is 0125. the value at 0125 is 37, and hence, the instruction LDA (BETA,X) loads the value 37 into the accumulator. This form of addressing is shown in the following illustration.



INDIRECT INDEXED

Another mode of indirect addressing uses index register Y and is illustrated by:

LDA (GAMMA),Y

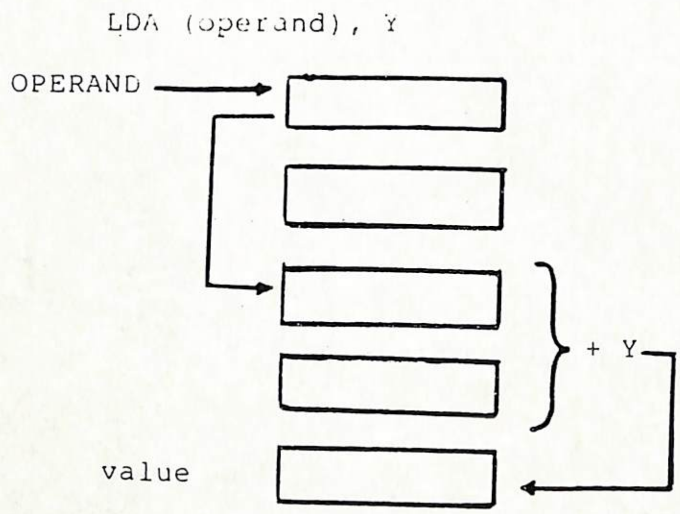
In this case, GAMMA references a page zero location at which an address is to be found. The value in index Y is added to that address to compute the actual address of the OPERAND. Suppose for example that:

GAMMA is 38 (hexadecimal)
Y contains 7
Locations 0039 and 0038 are 00 and 54
Location 005B contains 126

Then the address at 38 is 0054 and 7 is added to this giving an effective address 005B. The value at 005B is 126 which is loaded into the accumulator.

In indexed indirect, the index X is added to the OPERAND prior to the indirection. The indirect indexed, the indirection is done and then the index Y is added to compute the effective address. Indirect mode is always indexed except for a JMP instruction which allows an absolute indirect address as

exemplified by JMP (DELTA) which causes a branch to the address contained in locations DELTA and DELTA+1. The indexed indirect mode of addressing is shown in the following illustration.



ASSEMBLER DIRECTIVES

There are twelve directives used to control the assembly process, define values or initialize memory locations. Assembler directives always appear in the OPCODE field of an instruction and thus might be considered as assembly time OPCODES instead of execution time OPCODES. The directives are:

```
.IFE   .BYTE  .WORD  .LIB   .DBYTE  .OPT
.IFN   .PAGE  .SKIP  .FILE  * = or =     .END
```

All directives which are preceded by the period may be abbreviated to the period and three characters if desired eg, .BYT.

.BYTE is used to reserve one byte of memory and load it with a value. The directive may contain multiple OPERANDS which will store values in consecutive bytes. ASCII strings may also be generated by enclosing the string with quotes.

```
HERE     .BYTE 2
THERE    .BYTE 1, $F, @3, %101, 7
ASCII    .BYTE     'ABCDEFH'
```

Note that numbers may be represented in the most convenient form. In general, any valid MCS650X expression which can be resolved to eight bits may be used in this directive. If it is desired to include a quote in an ASCII string, this may be done by inserting two quotes in the string;

```
.BYTE   'JIM''S CYCLE'
```

could be used to print:

```
JIM'S CYCLE
```

.WORD is used to reserve and load two bytes of data at a time. Any valid expression, except for ASCII strings, may be used in the OPERAND field.

```
HERE      .WORD  2
THERE     .WORD  1, $FF03, @3
WHERE     .WORD  HERE, THERE
```

The most common use for .WORD is to generate addresses as shown in the above example labelled "WHERE" which stores the 16 bit addresses of "HERE" and "THERE". Addresses in the MCS-650X are fetched from memory in the order low-byte and high-byte, therefore, .WORD generates the values in this order. The hexadecimal port ion of the example ($FF03) would be stored 03,FF. If this order is not desired, use .DBYTE rather than .WORD.

.DBYTE is exactly like .WORD except the bytes are stored in high-byte, low-byte order.

```
.DBYTE     $FF03
```

will generate FF,03. Thus, fields generated by .DBYTE may not

be used as indirect addresses.

Equal (=) is the EQUATE directive and is used to reserve memory locations, reset the program counter (*), or assign a value to a symbol.

```
HERE      *=*+1          reserve one byte
WHERE     *=*+2          reserve two bytes
*=$200                   set program counter
NB=8                     assign value
MB=NB+%101               assign value
```

The = directive is very powerful and can be used for a wide variety of purposes.

Expressions must not contain forward references or they will be flagged as an error.

Example: * = C + D - E + F

would be legal if C, D, E and F are all defined, but would be illegal if any of the variables were a forward reference. Note also that expressions are evaluated in strict left to right order.

.PAGE is used to cause an immediate jump to top of page and may also be used to generate or reset the title printed at the top of page.

```
.PAGE              'THIS IS A TITLE'
.PAGE
.PAGE              'NEW TITLE'
```

If a title is defined, it will be printed at the top of each page until it is redefined or cleared. A title may be cleared with:

```
.PAGE          ' '.
```

.SKIP is used to generate blank lines in a listing. The directive will not appear but its position may be found in a listing, since it is treated as a valid input "card" and the card number printed on the left side of the listing will jump by two when the next line is printed.

```
.SKIP     2          skip two blank lines
.SKIP     3*2-1      skip five lines
.SKIP                skip one line
```

.OPT is the most powerful directive and is used to control the generation of output fields, listings and expansion of ASCII strings in .BYTE directives.

```
.OPT      ERRORS, LIST, MEMORY, GENERATE
.OPT      NOE, NOL, NOM, NOG
```

Also valid is:

```
.OPT      ERR
```

Default settings are:

.OPT MEM, LIST, ERR, NOGEN

Here are descriptions for each of the OPERANDS:

RORS [NO ERRORS]:
Used to control creation of a separate error file. The error file contains the source line in error and the error message. This facility is normally of greatest use to time-sharing users who have limited print capacity. The error file may be turned on and examined until all errors have been corrected. This listing file may then be examined. Another possibility is to run with:

```
.OPT        ERRORS, NOLISTING
```

until all errors have been corrected, and then make one more run with:

```
.OPT        NOERRORS, LISTING
```

LIST [NOLIST]:
Used to control the generation of the listing file which contains source input, errors and warnings, code generation, symbol table and instruction count if enabled.

MEMORY [NOMEMORY]:
Used to control generation of the memory file, which is used as an interface between the assembler and the simulator and various loader programs. The memory file contains information about symbols, line numbers and code generation, and is described in detail elsewhere in this document.

GENERATE [NOGENERATE]:
Used to control printing of ASCII strings in the .BYTE directive. The first two characters will always be printed, and subsequent characters will be printed (normally two bytes per line), if GENERATE is used.

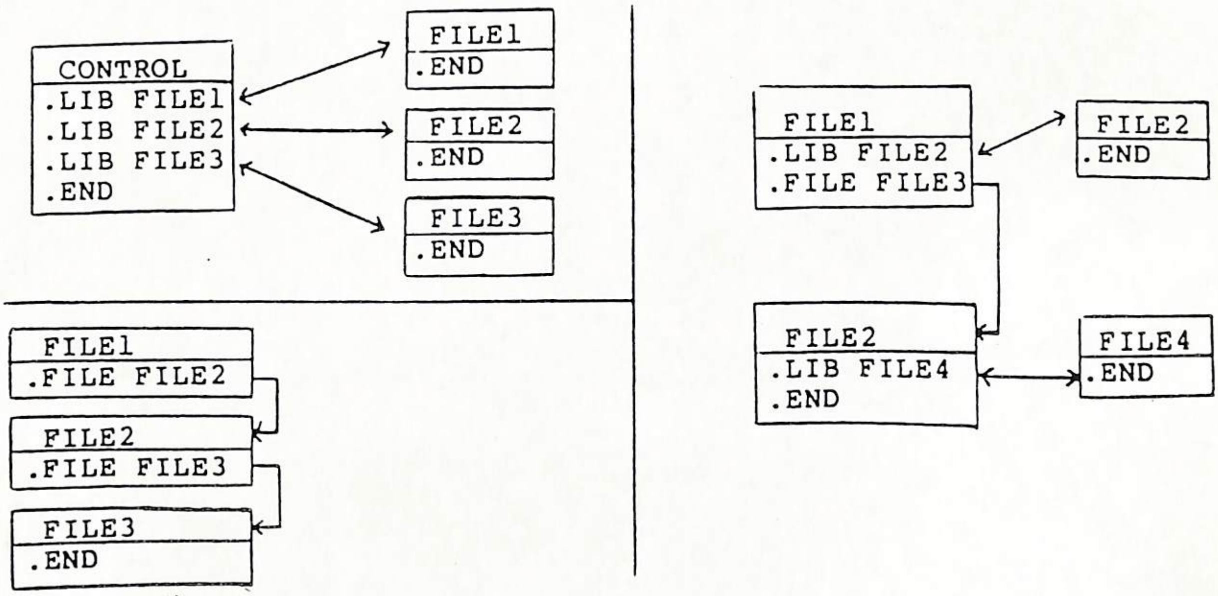
.END should be the last statement in a file and is used to signal the physical end of the file. Its use is optional, but highly recommended for program documentation.
NOTE : The symbol table will not be generated until a .END is found.

.LIB directive allows the user to construct control files containing this directive which inserts the module named into the assembly. When the assembler encounters this directive, it temporarily ceases reading source code from the file containing it and starts reading from the file named on the .LIB. Processing of the original source file resumes when end-of-file (EOF) or .END is encountered in the library file. The control file containing the .LIB can contain other assembler directives to turn the listing function on and off etc..

A library file called by a LIB may not contain another .LIB, but it may contain a .FIL.

.FIL terminates assembly of the file containing it and transfers source reading to the file named in the OPERAND. There are no restrictions on the number of files which may be linked by .FIL directives. Caution should be exercised when using this directive to ensure that no circular linkages are created. An assembler pass can only be terminated by (EOF) or .END directive.

The following is an example of .LIB control and .FIL control. This example is extensive and it should not be taken verbatum but used as a source of ideas.



.IFE / .IFN
Conditional Assembly Directives, these allow the same source file to be used to generate different versions of the final code, for different machines. The syntax is the same for both :-

```
          .IFE   expression   >
                 Code to be assembled
          <
```

In this case IF expression equals zero assemble the code between the > and <. The code to be assembled must start on a new line after the <IF, the < must be the first symbol on a line.

.IFN will cause the code between the > and < to be assembled if expr̶e̶s̶sion is not zero.

```
Eg.     MEM = 8
        COLS = 40
        * = $3B00     ;DEFAULT START ADDRESS
        WIDTH = 40    ;DEFAULT SCREEN WIDTH
        IFE  MEM-8  >
        * = $1B00     ;START FOR 8K PET
        <
        .IFN COLS-40  >
        WIDTH = 80    ;80 COLUMNS ON SCREEN
        <
```

In this example the program counter (*) would be set to $1B00 and width would be set at 40 (cols-40 = 0)

OUTPUT FILES

There are three output files generated by the assembler. Each file is optional through the use of the .OPT assembler directive. The listing file contains the program list, and symbol table. The error file contains all error lines and errors. The error file is included in the listing file. The interface file contains the object code for the loader.

LISTING FILE

The listing file will be produced unless the NOLIST option is used on the .OPT assembler directive. This file is made up of two sections: Program, and Symbol Table.

Program

This listing will always be produced unless the NOLIST option is selected. It contains the source statements of the program along with the assembled code. Errors and warnings appear after erroneous statements. An explanation of error codes are presented in part B. A count of the errors and warnings found during the assembly are presented at the end of the program.

Symbol Table

The symbol table will always be produced unless the NOSYM option is used. It contains a list of all symbols used in the program, and their addresses.

INTERFACE FILE

The format for the first and all succeeding records, except for the last record, is as follows:

; n1n0 a3a2a1a0 (d1d0)1 (d1d0)2 x3x2x1x0

Where the following statements apply:

1. All characters (n,a,d,x) are the ASCII characters 0 through F, each representing a hexadecimal digit.

2. ; is a record mark indicating the start of a record.

3. n1n0 = the number of bytes of data in this record. (in hexadecimal). Each pair of hexadecimal characters (d1d0) represents a single byte in the record.

4. a3a2a1a0 = the hexadecimal starting address for the record. a3 represents address bits 15 thru 12, etc.. The 8-bit byte represented by (d1d0)1 is stored in address a3a2a1a0; (d1d0)2 is stored in (a3a2a1a0) + 1, etc.

5. (d1d0) = two hexadecimal digits representing an 8-bit byte of data. (d1 = high-order 4 binary bits and d0 = low-order 4-bits). Maximum of 18 (Hex) or 24 (decimal) bytes of data per record is permitted.

6. x3x2x1x0 = record check sum. This is the hexadecimal sum of all characters in the record, including the n1n0 and a3a2a1a0, but excluding the record mark and the check sum of characters. To generate the check sum, each byte of data (represented by two ASCII characters) is treated as 8 binary bits. The binary sum of these 8-bit bytes is truncated to 16 binary bits (4 hexadecimal digits) and is then represented in the record as four ASCII characters (x3x2x1x0).

The format for the last record in a file is as follows:

; 00 c3c2c1c0 x3x2x1x0

1. ; 00 = zero bytes of data in this record. This identifies this as the final record in a file.

2. c3c2c1c0 = the total number of records (in hexadecimal) in this file, NOT including the last record.

3. x3x2x1x0 = check sum for this record.

Error messages are given in the program listing accompanying statement in error. The following is a list of all error messages which might be produced during assembly.

****A,X,Y,S,P RESERVED**

A label on a statement is one of the five reserved names (A,X,Y,S and P). They have special meaning to the assembler and therefore cannot be used as labels. Use of one of these names will cause this error message to be printed and zero bytes to be generated for the statement. The label does not get defined and will appear in the symbol table as an undefined variable. Reference to such a label elsewhere in the program will cause error messages to be printed as if the label were never declared.

HOW TO AVOID: Don't use A, X, Y, S or P as a label to a statement.

****.A MODE NOT ALLOWED**

Following a legal OPCODE, and one or more spaces, is the letter A followed by one or more spaces. The assembler is trying to use the accumulator (A = accumulator mode) as the OPERAND. However, the OPCODE in the statement is one which does not allow reference to the accumulator. Check for a statement labelled A (an illegal statement), which this statement is referencing. If you were trying to reference the accumulator, look up the valid OPERANDS for the OPCODE used.

****INVALID ADDRESS**

An address referred to in an instruction, or in one of the assembler directives (.BYTE, .DBYTE, .WORD), is invalid. In the case of an instruction, the OPERAND that is generated by the assembler must be greater than or equal to zero, and less than or equal to $FFFF (2 bytes long). (This excludes relative branches which are limited to +/- 127 from the next instruction.) If the OPERAND generates more than two bytes of code or is less than zero, this error message will be printed. For a .BYTE each OPERAND is limited to one byte, and for a .WORD each OPERAND is limited to two bytes. All must be greater than or equal to zero.

This validity is checked after the OPERAND is evaluated. Check for values of symbols used in the OPERAND field (see the symbol table for this information).

****FORWARD REFERENCE**

The expression on the right side of an equals sign contains a symbol that has not been defined previously. One of the operation of the assembler is to evaluate expressions or labels, and assign addresses or values to them. The assembler processes the input values sequentially, which means that all of the symbolic values that are encountered fall into two classes--already defined values and not previously encountered values. The assembler assigns defined values and builds a table of undefined values. When a previously used value is

discovered, it is substituted into the table and the assembler processes all of the input statements a second time using currently defined values.

A label or expression which uses a yet undefined value is considered to be referenced forward to the to-be-defined value.

To allow for conformity of evaluating expressions, this assembler allows for one level of forward reference so that the following code is allowed.

Card Sequence	Label	Opcode	Operand
100		BNE	New One
200	New One	LDA	#5

But the following is not allowed:

Card Sequence	Label	Opcode	Operand
100		BNE	New One
200	New One		Next + 5
300	Next	LDA	#5

This feature should not disturb the normal use of labels as the cure for this error

Card Sequence	Label	Opcode	Operand
100		BNE	New One
300	Next	LDA	#5
301	New One		Next + 5

is very simple and always solves the problem.

This error may also mean that the value on the right side of the = is not defined at all in the program; in which case, the cure is the same as for undefined values.

The assembler cannot process more than one computed forward reference. All expressions with symbols that appear on the right side of any equal sign must refer only to previously defined symbols for the equate to be processed.

****ILLEGAL OPERAND TYPE**

After finding an OPCODE that does not have an implied OPERAND, the assembler passes the OPERAND field (the next non-blank field following the OPCODE) and determines what type of OPERAND it is (indexed, absolute,,etc.). If the type of OPERAND found is not valid for the OPCODE, this error message will be printed.

Check to see what types of OPERANDS are allowed for the OPCODE and make sure the form of the OPERAND type is correct (see the section on addressing modes).

Check for the OPERAND field starting with a left paren. If it is supposed to be an indirect OPERAND, recheck the correct format for the two types available. If the format was wrong (missing right paren or index register), this error will be printed. Also check for missing or wrong index registers in an indexed OPERAND (form: expression, index register).

**IMPROPER OPCODE

The assembler searches a line until it finds the first non-blank character string. If this string is not one of the 56 OPCODES it assumes it is a label and places it in the symbol table. It then continues parsing for the next non-blank character string. If none are found, the next line will be read in and the assembly will continue. However, if a 2nd field is found it is assumed to be an OPCODE (since only one label is allowed per line). If this character string is not a valid OPCODE, the error message is printed.

This error can occur if OPCODES are misspelled, in which case the assembler will interpret the OPCODE as a label (if no label appears on the card). It will then try to assemble the next field as the OPCODE. If there is another field, this error will be printed.

Check for a misspelled OPCODE or for more than one label on a line.

**CAN'T EVAL EXPRESSION

In evaluating an expression, the assembler found a character it couldn't interpret as being part of a valid expression. This can happen if the field following an OPCODE contains special characters not valid within expressions (eg parentheses). Check the OPERAND field and make sure only valid special characters are within a field (between commas).

**INDEX MUST BE X OR Y

After finding a valid OPCODE, the assembler looks for the OPERAND. In this case, the first character in the OPERAND field is a left paren. The assembler interprets the next field as an indirect address which, with the exception of the jump statement, must be indexed by one of the index registers, X or Y. In the erroneous case, the character that the assembler was trying to interpret as an index register was not X or Y and the error was printed.

Check for the OPERAND field starting with a left paren. If it is supposed to be an indirect OPERAND, recheck the correct format for the two types available. If the format was wrong (missing right paren or index register), This error will be printed. Also check for missing or wrong index registers in an indexed OPERAND (form: expression, index register).

**LABEL START NEED A-Z

The first non-blank field is not a valid OPCODE. Therefore, the assembler tried to interpret it as a label. However, the first character of the field does not begin with an alphabetic character and the error message is printed.

Check for an unlabelled statement with only an OPERAND field that does start with with a special character. Also check for illegal label instruction.

**LABEL TOO LONG

All symbols are limited to six characters in length. When parsing, the assembler looks for one of the separating characters to find the end of a label or string. If other than one of these separators is used, the error message will be printed providing that the illegal separator causes the symbol to extend beyond six characters in length. Check for no spacing between labels and OPCODES. Also check for a comment card with a long first word that doesn't begin with a semicolon. In this case the assembler is trying to interpret part of the comment as a label.

**NON-ALPHANUMERIC

Labels are made up of from one to six alphanumeric digits. The label field must be separated from the OPCODE field by one or more blanks. If a special character or other separator is between the label and the OPCODE, this error message might be printed.

The 56 valid OPCODES are each three alphabetic characters. They must be separated from the OPERAND field (if one is necessary) by one or more blanks. If the OPCODE ends with a special character (such as a comma), this error message will be printed.

In the case of a lone label or an OPCODE that needs no OPERAND, they can be followed directly by a semicolon to denote the rest of the card as a comment (use of a semicolon tabs the comment out to the next tab position).

**DUPLICATE SYMBOL

The first field on the card is not an OPCODE so it is interpreted as a label. If the current line is the first line in which that symbol appears as a label (or on the left side of an equals sign), it is put in the symbol table and tagged as defined in that line. However, if the symbol has appeared as a label, or on the left of an equate, prior to the current line, the assembler finds the label already in the symbol table. The assembler does not allow redefinitions of symbols and will, in this case, print the error message.

**INDIRECT OUT OF RANGE

An indirect address is recognized by the assembler by the parentheses that surround it. If the field following an OPCODE has parens around it, the assembler will try to assemble it as an indirect address. If the OPERAND field extends into absolute (is larger than 255 - one byte), this error message will be printed.

This error will only occur if the OPERAND field is in correct form (ie an index register following the address), and the address field is out of page zero. To correct this, the address field must refer to page zero memory.

**PC NEGATIVE-RESET 0

An assembled program is loaded into core from position 0 to 64K (65535). This is the extent of the machine. Instructions can only refer to up to 2 bytes of information. Because there is no such a thing as negative memory, an attempt to reference a negative position will cause this error and the program counter (or pointer to the current memory location), will be reset to 0.

When this error occurs, the assembler continues assembling the code with the new value of the program counter. This could cause multiple bytes to be assembled into the same locations. Therefore, care should be taken to keep the program counter within the proper limits.

**RAN OFF END OF CARD

This error message will occur if the assembler is looking for a needed field and runs off the end of the card (or line image) before the field is found. The following should be checked for: a valid OPCODE field without an OPERAND field on the same card: an OPCODE that was thought to take an implied OPERAND, which in fact needed an OPERAND: an ASCII string that is missing the closing quote (make sure any embedded quotes are doubled - to have a quote in the string at the end, there must be 3 quotes - 2 for the embedded quote and one to close off the string); a comma at the end of the OPERAND field indicates there are more OPERANDS to come: if there aren't other OPERANDS, the assembler will run off the card looking for them.

**BRANCH OUT OF RANGE

All of the branch instructions (excluding the two jumps), are assembled into 2 bytes of code. One byte is for the OPCODE and the other for the address to branch to. To allow a forward or backward branch, this branch is taken relative to the beginning of the next instruction, according to the address byte. If the value of the byte is 0-127 the branch is forward; if the value is 128-255 the branch is backward. (A negative branch is in 2's complement form). Therefore, a branch instruction can only branch forward or backward 127 bytes relative to the beginning of the next instruction. If an attempt is made to branch further than these limits, the error message will be printed.

**UNDEFINED DIRECTIVE

All assembler directives begin with a period. If a period is the first character in a non-blank field the assembler interprets the following character string as a directive. If the character string that follows is not a valid assembler directive, this error message will be printed.

Check for a misspelled directive, or a period at the beginning of a field that is not a directive.

**UNDEFINED SYMBOL

This error is generated by the second pass. If, in the first pass the assembler finds a symbol in the OPERAND field (the field following the OPCODE or an equals sign), that has not been defined yet, that symbol is flagged for interpretation by pass two. If the symbol is defined (shows up on the left of an equate or as the first non-blank field in a statement), pass one will define it and enter it in the symbol table. Then a symbol in an OPERAND field before the definition will be defined with a value when pass two assembles it. In this case the assembly process can be completed.

However, if pass one doesn't find the symbol as a label or on the left of an equate, it never enters it in the symbol table as a defined symbol. When pass two tries to interpret the OPERAND field this type of symbol is in, there is no value for the symbol and the field cannot be interpreted. Therefore, the error message is printed with no value for the OPERAND.

This error will also occur if a saved symbol A, X, Y, S or P, is used as a label and referred to elsewhere in the program. On the statement that references the saved symbol, the assembler sees it as a symbol that has not been defined.

Check for use of saved symbols, misspelled labels or missing labels to correct this error.

NOTE:

When the assembler finds an expression (whether it is in an OPERAND field or on the right of an equals sign) it tries to evaluate the expression. If there is a symbol within the expression that hasn't been defined yet, the assembler will flag it as a forward reference and wait to evaluate it in the second pass. If the expression is on the right side of an equal sign, the forward reference is a severe error and will be flagged as such. However, if the expression is in an OPERAND field of a valid OPCODE, the first pass will set aside 2 bytes for the value of the expression and flag it as a forward reference. When the 2nd pass fills in the value of the expression, and the value of the expression is one byte long ie >256, the instruction is one byte longer than required, because the forward reference to page zero memory wastes one byte of memory - the extra one that was saved, because, during the first pass, the assembler didn't know how large the value was, so had to save for the largest value - two bytes.

COMMODORE 64 MONITORS

(XVM4.8 and XVM4.C)

The Commodore 64 monitors are essentially the same as VICMON except for the following considerations:

o To load the appropriate monitor enter:

```
LOAD "XVM4.8",8,1  (monitor at $8000)  SYS 32768   or
LOAD "XVM4.C",8,1  (monitor at $C000)  SYS 49152
```

o There is no WALK command.

o There is no TRACE command.

o In order to protect page zero from destruction, you must execute an E (enable) command before branching to your code with a G command. For example:

  Your code is constructed at C100. Before executing a G C100 command, type E C000 (or any other safe area of memory to save page zero). The monitor will use the specified location as a temporary storage area for page 0. When a B (breakpoint) command is encountered in your code, the monitor will restore page 0.

o After exiting from the monitor, perform a cold start by entering:

  SYS 64738

  Most programs will remain untouched.

THE DOS WEDGE
(DOS 5.1)

This program is an aid to the Commodore 64 user. When the program is operating, the user may enter disk commands and interrogate the error channel without using BASIC commands.

When the wedge program is running it is "wedged" into the operating system and BASIC interpreter. Thus the program can trap keyboard input before BASIC can see it.

The following is a list of DOS 5.1 commands:

>I0	Initialize drive 0.
>S0:S*	Scratch all files on drive 0 that start with the letter "S".
>$0	Read the directory on drive 0 and print it to the screen. (This does not destroy programs in memory.)
>$0:BASIC*	Read the directory on drive 0 and search for filenames starting with BASIC.
>(RETURN)	Interrogate the error channel and print error message on screen.
>#9	Change to device 9 etc.
>Q	Exit DOS wedge.
/ASM.C64	Load the program named "ASM.C64" (LOAD "ASM.C64",8).
↑ASM.C64	Load then run the program named "ASM.C64.
%PADDLE2	Load the program named "PADDLE2" without relocating. (LOAD "PADDLE2",8,1).
←PADDLE2	Save the program named "PADDLE2".

The following list shows differences between old versions of the DOS wedge and DOS 5.1:

- % command loads without relocating.
- Commands can be used in a program.
- DOS 5.1 pulls filename from quotes and ignores the rest of line.
- Commands do not have to start in column 1.
- DOS 5.1 does not auto relocate.
- Arrow left is save command.
- >#9 changes to device 9, etc.
- >Q exits DOS 5.1.

UNIVERSAL WEDGE

This program is an aid to the disk user. When the program is operating, the user may enter disk commands and interrogate the disk error channel without using BASIC commands.

When the wedge program is running it is 'wedged' into the operating system and BASIC interpreter. Thus the program can trap keyboard input before BASIC sees it. This is done by linking into the CHRGET routine in page zero.

The operation of the wedge program is accomplished using three commands that are typed in the first character position of a line. The greater-than symbol (>) is used to print the directories, send commands and read the error channel. The following examples illustrate the usage of the > command:

>I0	Initialize drive 0
>S1:S*	Scratch all files on drive one that start with the letter S
>$1	Read the directory on drive 1 and print it to the PET screen
> $0:BASIC*	Read the directory on drive 0 and search for filenames starting with the string 'BASIC'

The most notable attribute about this method of reading the directory is that it does not destroy programs in memory so you may examine the directory at any time.

The third use of the > command is interrogation of the disk error channel. This is done by typing:

>(RETURN)

the computer responds by printing the error message on the screen. This does the same thing as the following BASIC program:

```
10  OPEN15,8,15
20  INPUT#15,A,B$,C,D
30  PRINTA;B$;C;D
```

NOTE: After a cold start, do not attempt to read the error channel before sending a command.

Business keyboard users may substitute the @ (commercial at) symbol in place of >.

The second command is the slash command. It performs the same function as the BASIC 'LOAD' with a simplified format:

/ASSEMBLER

The above command loads the program named 'ASSEMBLER'. This does the same thing as the following BASIC command.

LOAD"ASSEMBLER",8

The third program is the LOAD/RUN command. This command is implemented using the up-arrow key ↑.

↑HURKLE

This example would load then run the program HURKLE.

The wedge program is the first program on the development disk, thus it may be loaded with the LOAD"*",8. This command causes an automatic initialization on Drive 0 when executed as the first disk command after a cold start.

INSTRUCTIONS FOR THE MINI-EDITOR

The MINI-EDITOR is included on the diskette supplied with this manual. There are two versions; EDITOR16K and EDITOR32K, used in 16K and 32K machines respectively. The only difference between the two versions is the load point.

The editor is used to enter and modify source files for the assembler. The editor retains all of the features of the screen editor plus; automatic line numbering, find, change, delete with range, and renumber. Other commands include get, put, break, kill, and format. All of the commands are detailed in the summary at the end of this appendix.

This is a line number oriented editor but with the functions of the screen editor it can be considered to have a character mode which includes the lines appearing on the screen. The editor commands operate in a similar fashion to the commands already existing in the computer's BASIC. There are several example files included on the diskette with the editor. Users would be well advised to try out the editor on these files in order to familiarize themselves with the commands.

The data files on which the assembler operates are ASCII characters with each line terminated by a carriage return. The only restriction on data files is in naming. Due to the method in which the assembler parses, you are not allowed spaces in filenames. The files are sequential and must be terminated by a zero byte $00.

LOADING THE MINI-EDITOR

The editor can be loaded with the wedge load command or with a BASIC load command:

```
/EDITOR32K    or
LOAD"EDITOR16K",8
```

To initiate the editor use the SYS command. The editor is started with the following commands, SYS7*4096 for the 32K version, and SYS3*4096 for the 16K editor. After typing the SYS command the editor will respond with the message 'MINI-EDITOR V121679'. At this point type a NEW command to clear the text pointers. You are now ready to edit assembler source files.

OPERATION OF THE MINI-EDITOR

When the MINI-EDITOR is in operation any statement typed:

```
10 FOR I=1 TO 10
```

will not be tokenized. Thus, you cannot type a BASIC line with the editor turned on. To avoid the above problem you must disable the editor with the 'KILL' command or reset to start clean.

Source files are loaded with the 'GET' command. As the file is loaded the editor adds line numbers. The editor starts its numbering at 1000. After editing your file, ensure that the

last line is a .FILE or a .END assembler directive. Then save your file on disk with the 'PUT' command.

The repeat key is enabled by typing (return). All of the keys on the keyboard will repeat when held down for more than one-half of a second. The repeat function is still operational after the editor has been killed. To disable the repeat function type: SHIFT-RUN/STOP followed by RUN/STOP (for version 2.0 BASIC). Type LOAD (return) followed by RUN/STOP (for version 4.0).

MINI-EDITOR COMMANDS

Auto Line Numbering

The function of this command is to generate new line numbers for entry of source code. In order to enable the auto command type the following:

AUTO nl(return)

where nl is the increment between line numbers printed. To disable the auto function type the auto command without an increment.

Break Command

The BREAK command jumps to the ROM resident monitor. This command is executed by typing:

BREAK(return)

Change String

The change command allows the user to automatically locate and replace one string with another (multiple occurrences). This command is entered in the following format:

CHANGE/strl/str2/,nl-n2

/	Delimits the strl and str2 (any character not in either str)
strl	Search string
str2	Replacement string
,nl-n2	Range parameters. The format is the same as the BASIC LIST command. If omitted the whole file is searched.

COLD

Resets the PET to its power on conditions. The disk unit is not reset nor is PET memory between $0100 and $0400. Its purpose is to clear memory to allow the HI-LOADER (MID-LOADER) to be loaded.

Delete

A delete range function has been included to make deletion quicker. The format is the same as the BASIC list command:

```
DELETE n1-n2
```

Ensure that you use the range parameters, as leaving them out causes the entire file to be deleted.

Find String

The find string command is used to locate specific strings in text. Each occurrence of the string is printed on the CRT. You can halt the printing with the SPACE bar. Printing can then be continued with the SPACE bar or terminated with the STOP key. The format of the FIND command is:

```
FIND/str1/,n1-n2
```

/	Delimiter
str1	Search string
,n1-n2	Range parm. same as BASIC list command

Formatted Print

The format command is used to print the text file in tabbed format like the assembler. For this function to work correctly you must type mnemonics in column two, or one space from labels.

```
FORMAT  n1-n2
        n1-n2     Range parms of the same
                  format as LIST.
```

NOTE: This command has the same controls as find. For example, SPACE halts printing, SPACE restarts, and STOP quit.

GET Files

Input assembler text files from disk. This command is used to load source files into the editor and append to files already in memory.

```
GET "FILE-NAME",n1,n2,n3
        n1        Begins inputting source
                  at this line.
        n2        Device #, default is 8
        n3        Secondary address
                  default is 8
```

NOTE: GET starts numbering at 1000 by 10.

KILL Command

This command causes the editor to disengage. This does not disable the repeat function if it has been turned on prior to KILL. To restart the editor, type the same command used to start the editor.

Resequence Lines

The NUMBER function allows the user to renumber all or part of the file in memory.

```
NUMBER n1,n2,n3
       n1          Old start line number
       n2          New start line number
       n3          Step size for resequence
```

PUT Command

The PUT command outputs source files to the floppy for later assembly. PUT has the ability to output all or part of the memory resident file.

```
PUT "0:NAME",n1-n2,n3,n4
      0:        Drive number, for disk only
      NAME      Output file name
      n1        Starting line number
      n2        Ending   line number
      n3        Device #, default is 8
      n4        default is 8
```

If n1-n2,n3,n4 are left out, the whole file is outputted to the default device.

When using the device number parameter to PUT a file the syntax should should be altered to

```
PUT"0:NAME",n1-,n3,n4
```

This is because of the way in which the editor parses the commands.

MINI-EDITOR COMMAND SUMMARY

COMMAND	DESCRIPTION
AUTO nl	Starts automatic line numbering
AUTO	Shuts off auto
BREAK	Jump to the monitor
CHANGE/sl/s2/,nl-n2	Change string
CHANGE/sl/s2/	Change string no range
COLD	Reset the PET
DELETE nl-n2	Delete range
FIND/sl/,nl-n2	Find string
FIND/sl/	Find string no range
FORMAT nl-n2	Print formatted
GET"FILE",nl-n2,n3	Bring in text
GET"FILE"	Short form get
KILL	Disable the editor
LIST	List lines of text
NUMBER nl,n2,n3	Renumber text
PUT"0:FILE",nl-n2,n3,n4	Save text on disk
PUT"l:FILE"	Save text short form

OPERATION OF THE PET ASSEMBLER

The assembler loads as a BASIC program. To start the assembler, enter a RUN command. This works because the program starts with a SYS command which is set up in the assembler source. An example of how to start a program in this manner is given just before the appendices of this manual. When the assembler is run it will print a copyright notice and print the first user prompt.

The first user prompt is for an object filename. If no object file is desired, type a return for this prompt. If an object file is desired the response would be something like:

OBJECT FILE? 1:TESTOBJ

Note that a drive number must be specified before the filename.

The second prompt is for hard copy. A null response will produce output. You must type N for no, followed by a return to defeat the printed listing.

The hard copy may be directed to the IEEE port (device #4) or to the User Port. (User Port handshake is Centronics parallel; see the notes at the end of this appendix for details). The user determines which method by responding Y or N to the IEEE Printer question. A null response defaults to a yes.

The final prompt is for the source filename. When the user types the name of the file to be assembled followed by return, the assembler starts operation. If a null response is detected the assembler returns control to BASIC. This allows the user to correct mistakes.

Upon startup the assembler initializes both of the disk drives, opens the source file and starts the assembly. If an object file has been requested it is also opened.

There are several errors which may occur on a system level rather than an assembly level. These errors are caused by disk problems and user errors. They are generally easy to solve as is presented in the following examples.

The first is 'FILE NOT FOUND'; which is produced when one of the following occurs:

1. The source file was not found.
2. A .LIB specifies a nonexistant file.
3. A .FIL specifies a nonexistant file.

This error is of the human type: eg, the user has mistyped a filename or placed the wrong diskette into the floppy.

The second error is 'FILE EXISTS'. This error is produced when the object file named already exists on the drive specified. This error can be cured by scratching the old file or changing the diskette.

The third error is 'READ ERROR'. This error is a disk read

error. Please refer to the Disk user manual for a description of the errors and their causes.

Notes on Operation

When the assembler is running, operation may be halted by pressing the RUN/STOP key. This only halts the assembly process; operation may be terminated at this point by pressing B or T key. These keys return control to BASIC or TIM respectively. Pressing any other key continues the assembly process. This feature is useful for users without printers, as the screen listing can be examined during assembly.

The assembler can send the printed output to the User Port with a Centronics parallel handshake method. This allows the user to attach a printer other than a Commodore printer. The data is transferred byte parallel with two handshake lines, Data Strobe and Acknowledge (both active low). The data is placed on the port then the Data Strobe is pulled low 6 usecs later, starting the handshake sequence. The computer now waits for the acknowledge line to be pulled low by the printer. When this happens, the data strobe line is released and the handshake sequence is completed for one character. The following presents the interconnection for the interface. The user should keep cable lengths to a minimum, as there are no line drivers in the computer on the output lines. The user should also check the loading of the device that is being connected because the computer can only drive one standard TTL load.

Interconnection Table

PET		Description	Centronics
M	(cb2)	Data Strobe	1
C	(pa0)	Data 1	2
D	(pa1)	Data 2	3
E	(pa2)	Data 3	4
F	(pa3)	Data 4	5
H	(pa4)	Data 5	6
J	(pa5)	Data 6	7
K	(pa6)	Data 7	8
L	(pa7)	Data 8	9
B	(ca1)	Acknowledge	10
N	(gnd)	Ground	16
A	(gnd)	Ground	33

PET LOADERS

The PET Assembler produces portable output in an ASCII format that is not directly executable. This output must be LOADED so the program can be executed. This is the function of a Loader.

There are three versions of the Loader included on the development disk. Each version is positioned in a different area of RAM memory. This allows the user to load anywhere in RAM by using the correct loader. The following table shows the names, load points and run commands for each of the three loaders.

NAME	LOAD ADDRESS	RUN COMMAND
LOADER	$0400	RUN
MID-LOADER	$3000	SYS 3 *4096
HI-LOADER	$7000	SYS 7 *4096

The loaders are about 512 bytes long and all operate in the same manner. When activated, the loaders print a copyright notice and prompt the user for a load offset. The offset is used to place object code into an address range other than the one that it was assembled into. This allows the user to assemble for an area where there is no RAM and load into a RAM area. The object can then be programmed into EPROM etc..

The offset is a two byte hexadecimal address that is added to the program addresses. If the program address, plus the offset, is greater than $FFFF, the address wraps around through $0000. If no offset is required pressing the RETURN key defaults to a zero offset. The following examples show how offset works.

Address	Offset	Load Point
$0400	$0000	$0400
$3000	$0000	$3000
$9000	$D000	$2000
$E000	$4000	$2000

After the offset is entered, the loader will prompt the user for the object filename to be loaded. The loader will then initialize both drives, search for the file and start the load. As the data is loaded, the program will print the input data to the CRT. This is for user feedback only. When the load is completed the loader prints the message 'END OF LOAD' and returns to BASIC.

There are three errors that can happen during a load. Errors are considered fatal; the load is terminated, the object file is closed, and control is returned to BASIC if an error is encountered. The following is a list of possible errors, which should be self documenting.

BAD RECORD COUNT
NON-RAM LOAD
CHECKSUM ERROR

CAUTION: The HI-LOADER (MID-LOADER for 16K machines) and the MINI-EDITOR load into the same RAM area. You must cold start the computer before using these loaders!

EXTRAMON COMMANDS

SIMPLE ASSEMBLER

```
.A   2000 A9 12    LDA #$12
.A   2002 9D 00 80 STA $8000,X
.A   2005 DEX:GARBAGE
```

In the above example the user started assembly at 2000 hex. The first instruction was load a register with immediate 12 hex. In the second line the user did not need to type the 'A' and address. The simple assembler retyped the last entered line and prompts with the next address. To exit the assembler, type 'RETURN' after the address prompt. Syntax is the same as the disassembler output. A ':' can be used to terminate a line.

BREAK SET

```
.B 1000 00FF
```

The example sets a break at 1000 hex on the FF hex occurence of the instruction at 1000. Break set is used with the quick trace command (see later). A break set with count blank stops at the first occurence of the break address.

DISASSEMBLER

```
.D 2000
.,   2000 A9 12    LDA #$12
.,   2002 9D 00 80 STA $8000,X
.,   2005 AA       TAX
```

Disassembles to the end of memory starting at 2000 hex. The three bytes following the address may be modified by using the cursor keys to move and modify the bytes. Then hit 'RETURN' and the bytes in memory will be changed. Extramon will then disassemble that line again.

```
.D 2000 3000
```

This disassembles from 2000 to 3000

ENABLE STOP

```
.E
```

This allows an exit from machine programs. If keyboard interrupts are still operating the program may be stopped by pressing the STOP and '=' keys at the same time.

FILL MEMORY

```
.F 1000 1100 FF
```

This fills the memory from 1000 hex to 1100 hex with the byte FF hex.

GO RUN

.G

Go to the address in the PC register display and begin running code. All the registers will be replaced with the displayed values.

.G 1000

Go to address 1000 hex and begin running code.

HUNT MEMORY

.H C000 D000 'READ

This hunts through memory from C000 hex to D000 hex for the ascii string 'READ' and prints the addrhss where it is found. A maximum of 32 characters may be used.

.H C000 D000 20 D2 FF

Hunt memory from C000 hex to D000 hex for the sequence of bytes 20 D2 FF and print the address where it is found. A maximum of 32 bytes may be used. Hunt can be stopped with the STOP key.

INTEGRATE MEMORY

```
.I F000
.' F000 54 4F 4F 20 4D 41 4E 59TOO MANY
.' F008 20 46 49 4C 45 D3 46 49 FILESFI
```

This displays hex and ascii until the end of memory.

.I F000 F080

This displays hex and ascii from F000 hex to F080 hex.

LOAD FROM TAPE

.L

Load any program from tape #1

.L "RAM TEST"

Load from tape #1 the program named "RAM TEST"

.L "RAM TEST",02

Load from tape #2 the program named "RAM TEST". Beware that load with a file name breaks the IRQ saved by the monitor. Do not use the GO command after load. Exit to Basic and re-enter the monitor.

MEMORY DISPLAY

```
.M 0000 0080
.:  0000 00 01 02 03 04 05 06 07
.
.
.:  0080 08 09 0A 0B 0C 0D 0E 0F
```

This displays memory from 0000 hex to 0080 hex. The bytes following the address may be modified by editing and then typing a return.

NEW LOCATER

```
.N 7000 77FF 1000 0400 8000
.N 7000 77FF 1000 0400 8000 W
```

This relocates machine code from 7000 hex to 77FF hex to a new location at 1000 hex. New locater fixes all 3 byte instructions in the range 0400 hex to 8000 hex. The 'W' option will relocate .WORD tables only. The new locater will not move instructions of 00, so transfer the tables first, then zero tables in from the copy. New locater stops and disassembles on a bad OP code.

QUICK TRACE

```
.Q
.Q 1000
```

The first example begins trace at the address in the PC of the register display. The second begins at 1000 hex. Each instruction is executed as in the walk command, but no disassembly is shown. The break address is checked for break on the n'th occurence. The execution may be stopped by pressing the STOP and '=' keys at the same time.

REGISTER DISPLAY

```
.R
      PC   SR AC XR YR SP
.;   0000 01 02 03 04 05
```

This displays the register values saved when extramon was entered. The values may be changed with the edit followed by RETURN. Use this instruction to set up the PC value before single stepping with '.W'.

SAVE TO TAPE

```
.S "PROGRAM NAME",01,0800,0C80
```

Save to tape #1 from 0800 hex up to but not including 0C80 hex and call the program "PROGRAM NAME". Beware that save with a file name breaks the IRQ saved by the monitor. Do not use the GO command after save. Exit to Basic and re-enter extramon.

TRANSFER MEMORY

.T 1000 1100 5000

This transfers memory in the range 1000 hex to 1100 hex, and start storing it at address 5000 hex.

UNDO STOP KEYS

.U

This disables exit from machine language programs with the STOP and '=' keys.

WALK CODE

.W

Single step starting at address in register PC.

.W 1000

Single step starting at address 1000 hex. Walk will cause a single step to execute, and will disassemble the next instruction. You can control the speed with the following keys :-

'<'	for single step
'RVS'	for slow step
'SPACE'	for fast stepping.

EXIT TO BASIC

.X

This returns to Basic ready mode. The stack value saved when entered will be restored. Care should be taken that this value is the same as when the monitor was entered. A 'CLR' in Basic will fix any stack problems.

MONITOR INSTRUCTIONS

G - Go run

L - Load from tape

M - Memory display

R - Register display

S - Save to tape

X - Exit to Basic

EXTRAMON INSTRUCTIONS

A - Simple assembler

B - Break set

D - Disassembler

E - Enable stop keys

F - Fill memory

H - Hunt memory

I - Integrate memory

N - New locater

Q - Quick trace

T - Transfer memory

U - Undo stop keys

W - Walk code

SECTION VII

DEVELOPMENT AIDS

DOS Wedge	(DOS 5.1)
PET Emulator	(CBM64 TO PET)
Sound Editor	(SIDMON)
Sprite Editor	(SPED)
Character Editor	(CHRED)

PET EMULATOR

(CBM64 TO PET)

The program "CBM64 TO PET" is a short form emulator that relocates the screen and BASIC to the following locations:

Bottom of BASIC	$0400
Top of BASIC	$8000
Screen	$8000

Note: The STOP/RESTORE function will NOT bring the CBM64 back to it origina state after this program is run.

PREFACE

The **Commodore** PET EMULATOR software package allows you to execute programs that were originally designed for the Commodore PET computer on the new Commodore 64.

The PET EMULATOR modifies the CBM Model 64 so that it will operate identically to the 2.0 Basic PET 2001 in most respects. This modification consists of two parts: memory re-configuration and system interaction interpretation. The exact technical specifications of just how the PET EMULATOR operates are well outside the scope of this document, however the more important conceptual information of its operation are presented.

It is recommended that you read the entire document before trying to use the EMULATOR to ensure that your first expe ience will be a successful one.

USER CONVENTIONS

It is recommended that you familiarize yourself with the Commodore keyboard. Here is a brief description of certain keys and symbols, and their respective function in reference to the PET EMULATOR and this manual.

Up Arrow	The Up Arrow character, on the upper left corner of the keyboard, is used to load and execute program.
/ and %	These two keys are used to load a program into the computer's memory. A more definitive explanation will be presented later in this manual.
@ and Greater Than	These two keys are used interchangeably as the command prompt for the emulator.

TABLE OF CONTENTS

1.0 Getting Started

The operation of the PET **Emulator** is not only simple but really transparent to the user. It was designed by Commodore for one reason only: Software Compatability. It is our desire that you find the program useful and of real benefit. Please take a moment to make sure that you have the following equipment connected properly, as per the instructions in the Commodore 64 User Manual.

1. Commodore 64
2. Commodore 1541 Disk Drive

1.1 Loading the PET Emulator

On the diskette contained in the package there is a small program that will load and execute the PET Emulator, the Emulator itself, and several PET Public Domain programs for your enjoyment.

To load the PET Emulator type: **LOAD"*",8.** After the computer replies READY, type **RUN.** This will load and execute the Emulator. You will notice a message appears on the screen indicating that the Emulator is loaded and running.

2.0 Theory of Operation

Memory Configuration

The CBM 64, in normal operation, stores BASIC programs in the $0800 to $9fff memory range (HEX), with the screen stored at $0400 to $0800. The PET stores BASIC programs at $0400 to $7fff, with the screen stored at $8000 to $8fff.

The Emulator re-configures the Commodore 64 memory so that it duplicates the PET internally. Thus POKES to the screen, POKES to the program, and other such direct access operations work properly.

System Interaction Interpretation

Many PET programs access the system directly with PEEKS, POKES and WAITS. Most of the common PEEKS, POKES and WAITS are interpreted by the Emulator and should operate exactly as they would on the PET. These locations are as follows:

Location	PEEK	POKE	Operation
50003	x		BASIC version type test
59464	x	x	CB2 sound frequency
59467	x	x	CB2 sound on/off
59466	set to 15		
59468	x	x	Set upper/lower case

All POKES and PEEKS between $0000 and $03ff (when possible) are translated. **POKES not able to be interpreted return the message 'illegal quantity error'.**

Cassette buffer #2 is also available for machine code programs (same as the PET). Machine language programs that do not call system routines will work with no modifications if they reside in this cassette buffer.

The CB2 sound is emulated as closely as possible. Certain very high tones available on the PET can not be obtained on the 64, and the pitch of the tones varies across the scale. Musical tunes may not be emulated correctly, but other sound effects usually sound better under the emulator than they may have on the PET.

3.0 Operation

The Emulator loads into high memory on the 64, $C000 (HEX). It may be loaded directly by typing: LOAD"EMULATOR",8,1 and then after the computer replies READY, type SYS 12*4096. As stated above, the first program on the diskette is a program that will load and execute the Emulator for you. In addition this program contains the logic to allow screen color selections before actually loading the Emulator.

4.0 DOS 5.1

Included on the diskette, and operable with the Emulator, is another valuable product from Commodore, the DOS 5.1 Universal Disk Operating System program. This program is more commonly called the DOS Wedge because it 'wedges' itself in between the computer and the disk drive to facilitate disk operations such as LOAD and SAVE.

The DOS Wedge provides a very useful 'short cut' method of communicating with the disk drive. It cannot make the disk drive do anything more than can be done through commands in a BASIC program, but allows the user to communicate in a direct mode.

This version of the DOS Wedge will actually allow the 'short form' commands to be given from within a BASIC program as well.

4.1 Command Summary

Normal DOS Wedge Commands

up arrow	Load and Run a program
/	Load a program into the normal BASIC program area
%	Load a program into the area in memory dictated by the programs load address
left arrow	Save a program
@	Read and display the disk drive error channel
@$x	Display the directory of the disk in drive x (ie. @$0 for drive 0)
@$x:pgm*	Search the directory on drive x for the files that begin with 'pgm'
@n:name,id	Header a NEW disk, using 'name' and 'id'
@r0:a=b	Rename file 'b' on drive 0 to 'a'
@c0:a=0:b	Copy file 'b' on drive 0 to file 'a' on drive 0
@s0:name	Scratch file 'name' on drive 0
@uj	Reset disk drive
@#9	Change the active unit number from the default value of 8 to 9
@q	Quit, turn off, the DOS Wedge and the Emulator
@ix	Initialize disk drive x

File names for the LOAD and SAVE commands may be within quotes anywhere on a line of the screen (as in a directory listing), and all other extraneous information on the line is ignored.

Using the DOS Wedge from within a program:

```
10 INPUT "DO YOU WANT TO LIST THE DISK DIRECTORY";A$
20 IF A$="Y" GOTO 40
30 STOP
40 @"$0"
```

5.0 Combinations of Mode/Memory

New commands added to the DOS Wedge to support the Emulator include:

@m toggle memory configuration

@e toggle emulate mode (PET/64)

Either of the latter two commands will display the current mode/memory status. The @m command 'news', erases, the BASIC program currently in memory. Therefore, **'are you sure'** is asked before the command is executed.

The emulator normally loads up into the PET memory mode with the PET emulator on. Thus four combinations of mode/memory exist. When changing modes warnings are printed where potential confusion exists. Now once the computer is in PET memory mode with the Emulator on, the system operates as a PET as described above. When in 64 memory mode with the emulator off, the emulator has no effect on the normal operation of the Commodore 64.

Please note that when in the PET memory mode the RUN/STOP key is disabled.

SOUND EDITOR

(SIDMON)

SIDMON allows you to create sounds by editing the SID chip registers. (A detailed description of the SID chip can be found in Section II of this binder.) To load SIDMON enter:

```
LOAD "SIDMON",8
RUN
```

The following screen will appear:

```
MODE:                    *** SIDMON 1.3 ***

FREQUENCY           PULSE WIDTH
 5000                1024                                 (for voice 1)
 5000                2048                                 (   "      2)
 5000                3072                                 (   "      3)

ATK  DEC  SUS  RLS  GATE  SYNC  RING    ∧   ⟍   ⊓   *
 1    3    13   10    0     0     0     0   0   0   0    (for voice 1)
 1    3    13   10    0     0     0     0   0   0   0    (   "      2)
 1    3    13   10    0     0     0     0   0   0   0    (   "      3)
                                        1   2   3   4
FILTER:         RESONANCE: 0
 0
 0
 0                                      +-   UP/DN/ON/OFF

LOW  BAND  HIGH  CUT3                   D    DIVIDE BY TWO
 0    0     0     0
                                        M    MULTIPLY BY TWO
MASTER VOLUME: 15

CUTOFF FREQUENCY: 0

VOICE: (F1,F3,F5): 1
```

Commands are entered by pressing the reversed key (□) associated with that function. Up/down and on/off settings are controlled by pressing the "+" and "-" keys. This is best described by using examples:

First select your wave form (∧ ⟍ ⊓ *) by pressing 1,2,3 or 4.
Then, gate the sound ON by pressing "G" followed by "+".
Next you can alter the frequency by pressing "F", followed by:

- \+ to raise the frequency
- \- to lower the frequency
- M to multiply the frequency by 2
- D to divide the frequency by 2

Attack, decay, sustain, release, etc. can be altered in the same manner. Any one of the three voices can be selected by F1, F3, or F5.

After creating the desired sound, jot down the numbers in the registers for later use in your programs.

```
 SSSS   IIIIII  DDDDD    MMM      MMM    OOO    NNN   NN
SS  SS    II    DD  DD   MMMM    MMMM   OO OO   NNNN NN
SS        II    DD   DD  MM MM  MM MM  OO   OO  NN NNNN
 SSSS     II    DD   DD  MM  MMMM  MM  OO   OO  NN  NNN
    SS    II    DD   DD  MM   MM   MM  OO   OO  NN   NN
SS  SS    II    DD  DD   MM         MM   OO OO   NN   NN
 SSSS   IIIIII  DDDDD    MM         MM    OOO    NN   NN
```

Sound Interface Device MONitor

SIDMON

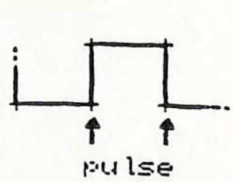
SIDMON is a program which allows you to create sounds using the 6581 Sound Interface Device in the Commodore 64. It is difficult to randomly try different sounds or keep poking around until you get the sound you want. SIDMON does all of the poking for you. You just look at the screen and change whatever you want. If you don't like the change, change it back. It's that simple to use!

The commands are usually the first letter of the register name. In some cases though, the letter has already been taken by another command, so the second letter is used instead. The proper letter to press is highlighted in reverse on the screen.

First, lets review the descriptions of the different registers...

FREQUENCY : This register sets the number of sound waves per second produced by the SID.

PULSE WIDTH : This register forms a number which linearly controls the duty cycle of the pulse waveform on the voice that is being worked on.

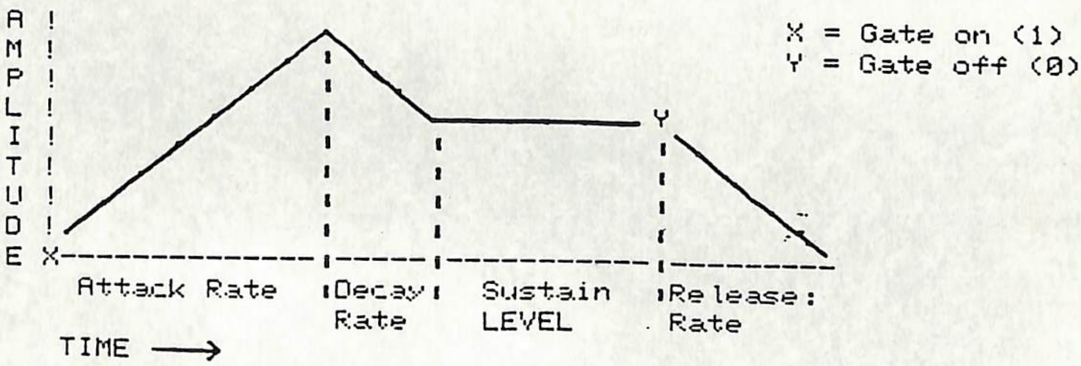


ATTACK : The attack rate determines how rapidly the output of the voice rises from zero to peak amplitude when the voice is gated.

DECAY : The decay cycle follows the attack cycle and the decay rate determines how rapidly the output falls from peak amplitude to the selected sustain level.

SUSTAIN : You can select 1 of 16 sustain levels for the voice. The sustain cycle follows the decay cycle and the voice will remain at the selected sustain level as long as the gate bit remains on (1).

RELEASE : There are 0 to 15 release levels of which one can be selected. The release cycle follows the sustain cycle when the gate bit is turned off (reset to zero). At this time, the output of the voice being worked on will fall from the sustain amplitude to zero amplitude at the selected release rate.



GATE : When the gate is set to a one, the attack/decay/sustain cycle begins. When the gate is reset to a zero, the release cycle begins. The gate must be set for the selected output of the voice to be audible.

SYNC : The sync, when set to a one, synchronizes the fundamental frequency of the voice being worked on with the frequency of voice 3. Varying the frequency of the present voice with respect to voice 3

produces a wide range of harmonic structures from the present voice at the frequency of voice 3. In order for sync to occur, voice 3 must be set to some frequency other than zero but preferably lower than the frequency of the present voice. No other parameters of voice 3 have any effect on sync.

RING : The ring register, when set to a one, replaces the triangle waveform of the present voice with a ring modulated combination of the present voice and voice 3. Varying the frequency of the present voice with respect to voice 3 produces a bell or gong-like sound.

/\ : The triangle waveform has a mellow, flute-like quality.

|\ : The sawtooth waveform is rich in even and odd harmonics and has a bright, brassy quality.

⊓ : The harmonic content of the pulse waveform can be adjusted by the pulse width register, producing tone qualities from a bright, hollow square wave to a nasal, reedy pulse.

* : This output is a random signal. The sound quality can be varied from a low rumbling to a hissing white noise via the frequency register.

FILTER : When set to a zero, the present voice appears directly at the audio output and the filter has no effect on the sound. If set to a one, the present voice will be processed through the filter and the harmonic content of the voice will be altered.

RESONANCE : This register controls the resonance of the filter. Resonance is a peaking effect which emphasizes frequency components at the cutoff frequency of the filter, causing a sharper sound. There are 16 resonance settings ranging linearly from no resonance (0) to maximum resonance (15).

LOW : When set to a one, the low pass output of the filter is selected and sent to the audio output. For a given filter input signal, all frequency components above the cutoff are attenuated at a rate of 12 dB/octave. The low pass mode produces full bodied sounds.

BAND : This is similar to LOW but, all frequency components above and below the cutoff are attenuated at a rate of 6 dB/octave. The band pass mode produces thin, open sound.

HIGH : This is the opposite of LOW, in that when set to a one, all frequency components below the cutoff are attenuated at a rate of 12dB/octave. This mode produces a tinny, buzzy sound.

CUT 3 : When set to a one, the output of voice 3 is disconnected from the direct audio path. Setting voice 3 to bypass the filter and setting cutoff 3 to a one prevents voice 3 from reaching the audio output. This allows voice 3 to be used for modulation purposes without any undesirable output.

MASTER VOLUME : Select one of 16 overall volume levels for the final composite audio output. The values this register can hold range from 0

(no sound) to 15 (max sound).

CUTOFF FREQUENCY : This register is set to contain a number which controls the cutoff (or center) frequency of the programable filter.

SIDMON COMMANDS:

There are several commands that are used in SIDMON. These commands, as mentioned before, are highlighted on the screen in reverse. When a valid key is pressed, that key is displayed next to "MODE:" on the top line. The only keys that will not be displayed are +, -, M and D. For these keys to have any effect, another letter must already be displayed next to the mode.

CLR HOME : To completly reset SIDMON press the <SHIFT> and <CLR HOME>. All of the values will be reset to their default. NOTE: ANY CHANGES YOU HAVE MADE WILL BE LOST!!!

F1, F3 and F5 : These three function keys are used to select the voice that you are currently working on. F1 is voice 1. F3 is voice 2 and F5 is voice 3. The voice that is presently being worked on is noted next to where it says "VOICE (F1,F3,F5):" on the bottom line of the screen.

+/- : The add and subtract keys are used to raise or lower values in the different registers. In some cases, such as "gate" which is either off or on, the plus (+) key is used to turn on and the minus (-) key is

used to turn off.

M/D : The "M" key and the "D" keys are used to multiply or divide respectivly. These keys work only on frequency and on pulse width.

F : The "F" key is used to select frequency. Follow this key with either +, -, M or D to change the frequency. While in frequency mode, the plus and minus keys add or subtract 16 from the frequency of the voice that you are working on. If you only want to add 8 to the frequency, multiply the frequency by two by using the "M" key. Then, using the "+" key add 16, then divide by 2. Your frequency should now be 8 more than when you started.

P : To select pulse width, press the "P" key. The pulse width mode is similar to the frequency mode. The only difference is that the plus and minus keys change the pulse width by 64. Remember, pulse width only applies when you are using square wave.

A, D, S and R : These keys select attack, decay, sustain and release, respectively. The values of each range from 0 to 15. Plus and minus change these registers by 1. The "M" and "D" keys do not work while in this mode.

G : This key selects gate. A one (+) in this register starts the attack, decay and sustain. When reset to a zero (-), the release sequence begins. This register must be manipulated for you to get any sound from the SID.

Y and I : These keys select sync and ring. On and off are the only 2 states these registers can have. The + key turns the register on and the - key turns it off.

0, 1, 2, 3 and 4 : Keys one thru four select the type of waveform that the voice you are working on will have. The "1" key selects triangle. "2" selects sawtooth. Pulse is chosen by pressing the "3" key and the "4" key selects noise. For pulse width to have any effect, the pulse waveform must be selected. Only one of these four choices may be on at any one time. The 0 key will turn off all waveforms.

T : This selects filter. As in gate, sync and ring, the filter has either an on or an off state.

E : Resonance can be set by pressing the "E" key. It has a range of 0 to 15, which can be modified by using the + and - keys.

V : SIDMON has a master volume which can be changed by pressing "V" followed by + or - keys.

U : The cutoff frequency is selected by pressing "U". This, followed by + or -, changes the cutoff frequency. (Divide and multiply does not work in this mode.)

Try to recreate this sample:

```
MODE:          *** SIDMON 1.4 ***
FREQUENCY. PULSE WIDTH
 4968         1024
 5000         2048
 5032         3136

ATK DEC SUS RLS GATE SYNC RING  ∧ ∧ ⊓ *
 1   3   13  10   1    0    0   . ! . .
 1   3   13  10   1    0    0   ! . . .
 1   3   13  10   1    0    0   . . ! .

FILTER:          RESONANCE: 0
 0
 0
 0                             UP/DN/ON/OFF

LOW  BAND HIGH CUT3        DIVIDE BY TWO
 0    0    0    0
                           MULTIPLY BY TWO
MASTER VOLUME: 15

CUTOFF FREQUENCY: 0

VOICE (F1,F3,F5): 1
```

For more information on the SID, see the Programmers' Reference Guide.

CHARACTER EDITOR

The characters that are printed onto the screen through PRINT statements (or POKE statements to screen memory) are stored in ROM. There are two such character sets, each occupying 2K (2048) bytes.

1) The usual GRAPHICS character set (POKE 53272,21)

2) The LOWER CASE character set (POKE 53272,23)

On the COMMODORE 64, it is possible to have character sets stored in RAM - 8 are theoretically possible - 5 are actually usable.

The RAM character sets are numbered 0-7. Each set occupies 2K of memory and is activated by poking register 24 of the 6566 Video Chip (i.e. memory location 53272) with an appropriate number.

CHARACTER SET	LOCATION	ENABLE WITH
0	0 - 2047	POKE 53272,17
1	2048 - 4095	POKE 53272,19
2	4096 - 6143	POKE 53272,21
3	6144 - 8191	POKE 53272,23
4	8192 - 10239	POKE 53272,25
5	10240 - 12287	POKE 53272,27
6	12288 - 14335	POKE 53272,29
7	14336 - 16383	POKE 53272,31

Note:

1) Character Set 0 is not available since this part of memory is used by the operating system and screen memory.

2) The pokes that enable Character Sets 2 and 3, in fact, enable the ROM character sets.

Thus the only character sets that are available for use in RAM are sets 1, 4, 5, 6, and 7. Since these sets are located in the same part of memory that a BASIC program normally occupies, it may be necessary to move the BASIC program up in memory whenever a RAM character set is used (See the instructions in the Sprite Editor to see how this is done). This will not always be necessary. For example, if an application program only requires 6-8K of memory, then Character Sets 5, 6, or 7 could be used, and could even be saved with the BASIC program itself - thus avoiding having to load the character set from within the program.

This CHARACTER EDITOR allows you to create, edit, etc. any of the five available character sets. Any set you create can be saved to disk to be used within your programs.

There are two distinct modes in this editor, each with its own set of commands - CHARACTER SELECTION MODE and EDIT MODE.

CHARACTER SELECTIONS MODE

This is the mode that the Character Editor starts with when it is first run. In this mode, a box cursor flashes over one of the 64 characters displayed near the bottom of the screen (whenever this box cursor flashes, you know that you are in CHARACTER SELECTION MODE and not in EDIT MODE).

The cursor can be moved from character to character using the normal cursor keys (up, down, left, and right). When the cursor is positioned over the character you wish to edit, press the RETURN key. That character will then be displayed on the EDIT GRID at the upper left of the screen ready for editing. The character will also be displayed to the right of the edit grid in each of the 16 colors.

While in Character Selection Mode, several commands are available. These are summarized in the following table.

Keystroke	Effect
CRSR-RT, LFT, -UP, DWN	Moves the box cursor to any of the 64 characters displayed on the screen.
RETURN	Selects the character under the cursor for editing. The character is displayed on the EDIT GRID and EDIT MODE is entered.
CTRL-N	Displays the NEXT 64 characters of the present character set at the bottom of the screen (There are 256 characters in each character set, but only 64 are displayed at any given time).
CTRL-B	Steps through each of the 16 BACKGROUND colors.
CTRL-E	Steps through each of the 16 EDGE (Border) colors.
Any of the # keys 1-7.	Displays the corresponding character set. (Note: Character Sets 2 and 3 can be displayed but since they are in ROM they cannot be edited.)
CTRL-L	LOADS a specified character set from disk into memory.
S	SAVES the character set, presently being edited, to disk. You are prompted for a filename and the number of the set you wish to save that set to. (Note: If you are presently editing Character Set 1, say, you can still save it to Character Set 4, 5, 6, or 7 if you wish.)
Q	QUIT the editor. Once you have quit the editor, you may restart it, with all character sets in tact, simply by typing RUN. The CHAR BOOT program does not have to be reloaded and run.

EDIT MODE

The commands available in Edit Mode are almost exactly the same as the commands available to the Sprite Editor.

The following commands are exactly the same (See the Sprite Editor's Instructions).

a) CRSR-RT, CRSR-LFT, CRSR-UP, CRSR-DWN.
b) SPACE, DEL, . , HOME, RETURN, CLR.
c) CTRL-R, F1, F2, F3, F4, &.
d) CTRL-B, CTRL-E.

The following commands are not implemented, simply because they do not apply to character sets.

a) CTRL-P, + , - , CTRL-D, CTRL-C.
b) > , CTRL-X, CTRL-Y, CTRL-V.

Note: S and CTRL-L are not implemented in Edit Mode, but are available in Character Selection Mode.

ADDITIONAL COMMANDS

a) CTRL-A — This allows the character being edited to be ASSIGNED to some character in the character set. When CTRL-A is pressed, the box cursor flashes over the character originally selected. The edited character can be assigned to this character or to another character by first using the cursor keys and/or CTRL-N to position the cursor over the desired character and then pressing RETURN.

b) Q — Pressing Q in Edit Mode allows you to re-enter Character Selection Mode without assigning the character to any character in the character set. The edited character, however, is lost.

NOTE: To use the Character Editor, the program CHAR BOOT must be loaded and run first.

SPRITE EDITOR

The data that defines a sprite on the Commodore 64 is stored in 64 consecutive bytes called a 'page' (this is not the usual page concept associated with the 6502 or 6510). Up to 256 sprite definitions are theoretically possible, but not all of these are available.

The sprite pages start at the beginning of memory, so for example, page 0 occupies memory locations 0-63, page 1 occupies locations 64-127, and so on. Page 32 begins at 32*64 = 2048 and extends for 64 bytes, page 150 begins at 150*64 = 9600, and so on.

Since the operating system uses most of the memory locations from 0-2048 for its own use, several pages here will not be available for sprite definitions. Others are not available as well.

Here is what is available and what is not.

Sprite Pages	Memory Locations	Status	Reason
0-12	0-831	Not Available	Operating system
13-15	832-1023	Available	Cassette buffer
16-31	1024-2047	Not Available	Screen Memory
32-63	2048-4095 (2K)	Available	See below
64-127	4096-8191 (4K)	Not Available	I don't know why!!
128-255	8192-16383 (8K)	Available	See below

Normally pages 32-63 (and some in the range 128-255) would not be available for sprite definitions since the BASIC program is stored there. However, it is easy to move a BASIC program up in memory to wherever you wish. Thus this area can be considered as available for use.

For example, to move a BASIC program up so that it starts at 16384, and hence make available ALL of the room above, the following steps are required:

1) POKE 16*1024,0 (Note: 16*1024=16384)
2) POKE 44,16*4
3) POKE 43,1 (Normally this location is 1, so this step may not be required.)
4) LOAD "YOUR PROGRAM" as usual.

These steps could be done in a small boot program, such as the SPRITE BOOT program, so that the user will not have to be concerned with such details.

This Sprite editor allows you to create, edit, save, etc. sprite definitions on any of these pages. A sprite is created or edited on a 24 X 21 grid (matrix) using the following commands:

Keystroke	Effect
CRSR-RT	Move the cursor (indicated in reverse field) one location to the right. When at the extreme right end of a line, the cursor will wrap around to the beginning of the same line.
CRSR-LFT	Move the cursor position one location to the left. Wrap around can occur to the right of the same line.
CRSR-UP	Move the cursor one line up. Wrap around can occur to the bottom of the same column.
CRSR-DWN	Move the cursor one line down. Wrap around can occur to the top of the same column.
SPACE	Erase any point plotted at the current cursor position.
DEL	Erase any point plotted at the previous cursor position.

Note: These six commands are supported by the repeat key.

.	Plot a point at the current cursor position.
HOME	Move the cursor to the top left corner of the grid.
RETURN	Move the cursor to the beginning of the next line. Wrap around can occur to the top (home) position.
CLR	Erases all points on the grid.
CTRL-R	Reverses the entire grid.
F1	Move the entire grid up one line.
F2	Move the entire grid down one line.
F3	Move the entire grid left one column.
F4	Move the entire grid right one column.
£	Rotates the sprite 90 degrees. Note: Rotating a 24 X 21 sprite can cause part of the sprite to disappear (since it is not square). The part that disappears is in fact preserved in a buffer, as can be seen by pressing the pound key once again - the sprite should now be upside down. Pressing it twice in succession again will bring it back to its original position - in fact, pressing RETURN at any time will put you back into EDIT mode. Whatever you see on the screen will be stored in the sprite definition and anything else in the buffer will be lost.

CTRL-P	Prompts for a new page number which is then displayed for editing purposes.
+	Display the sprite on the next page for editing.
-	Display the sprite on the previous page for editing.
CNTL-D	Display a range of sprites for viewing only.
S	Save a range of sprite definitions to disk.
CTRL-L	Load a previously stored table of sprites into memory.
CTRL-C	Copy a range of sprite definitions from one area of memory to another.
CTRL-B	Steps through each of the 16 Background colours.
CTRL-E	Steps through each of the 16 Edge (Border) colours.
>	Steps through each of the 16 sprite colours.
CTRL-X	Expand/contract the sprite being edited in the horizontal direction (acts like a toggle).
CTRL-Y	Expand/contract the sprite being edited in the vertical direction (acts like a toggle too).
CTRL-V	View the sprite, currently being edited, moving randomly on a clear screen.

PRESSING
- SPACE : stops the motion (press SPACE again to restart it).
- RETURN : returns you to the EDITOR.
- CTRL-B, CTRL-E, CTRL-X, CTRL-Y, and > have the same effect as above.
- \+ : speeds up the motion.
- \- : slows down the motion.

Q	Quits the EDITOR. (Once you have quit the EDITOR, you can restart it simply by typing RUN. You do not have to re-load the BOOT program.)

Note: To use the SPRITE EDITOR, you must load and run the SPRITE BOOT program first.

SPRITE EDITOR
(SPED)

INTRODUCTION

This program allows you to easily create and modify sprites on the Commodore 64. Many features to allow easy manipulation of sprites are included, as well as several commands to make the sprites created by this program usable by other programs.

SCREEN FORMAT

The screeen is basically divided into 4 areas by the sprite editor program. The first is the large sprite creation box. Here the sprite is formed using the "*" and " " symbols. The horizontal axis of the box is lettered from 0 to 23, giving 24 possible bit positions. The vertical axis is numbered from 0 to 20, giving 21 possible bit positions. In this box, an entire sprite is represented.

The second area is the information box. Current sprite parameters are displayed here. Also, when a command needs a parameter to operate, a cursor will appear at the appropriate place in this box.

The third area is the strip below the creation box. Here is where the sprite currently being edited normally appears.

The fourth area contains the command menu, where most of the commands are displayed. However, the menu in the area under the information box can be replaced by the current sprite by the F1 key. This is especially useful when working with expanded sprites.

General Notes: The first letter of a command is enough. When a command requires an input (like sprite number), a cursor will appear in the information box. Type the answer there, ending it with a return. If you type an incorrect letter, use the delete key. If your response is illegal, the program will ignore it, keeping the old value of the parameter.

THE INFORMATION BOX

The information box contains the following information:

	PARAMETER	POSSIBLE VALUES
1.	Sprite number	0-47
2.	Name of current file	any 5 letters
3.	Assumed address	any 4 digit hex number
4.	Range	0-47
5.	Type of sprite displayed	HIRES or MULTI
6.	Foreground color	any of the 16 colors
7.	Multi-color register 0	any of the 16 colors
8.	Multi-color register 1	any of the 16 colors
9.	X expand	YES or NO
10.	Y expand	YES or NO
11.	X position	0-319
12.	Y position	0-199

13. The bottom blank line of the information box is used for other command inputs that don't need to be continually displayed.

COMMANDS

1. E(dit) : requests a sprite number from 0 to 47. That sprite becomes the current sprite. The current appearance of the sprite is displayed both in the creation box and the current display area.
2. N(ext sprite) : selects the next sprite as the current sprite.
3. T(ype) : selects between displaying the current sprite as a hires (h) or a multi-color (m) sprite.
4. M(ove here) : asks for a sprite number. That sprite is copied into the current sprite. Note that this operation destroys the previous sprite.
5. O(r) : asks for a sprite number. That sprite is ORed into the current sprite.
6. C(olor) : allows choice of colors for the sprites. A cursor will appear at each of the color parameters in turn. Type the three letter abbreviation for the color of your choice for each register in turn.
7. X(pand) : selects expansion in the x and y directions. A cursor will appear at the XEXP and YEXP positions on the information box. Answer YES or NO (or Y or N).
8. P(osition) : allows specification of the screen position of the current sprite.
9. R(ange) : sets a range of sprites for use with the SAVE, LOAD, BYTE, and DISPLAY commands. The form is ##.## (i.e. 0:12), for first and last sprites to be affected by an operation.
10. D(isplay) : animation command. A time is requested. Answer with a number under 1000. Just hitting return allows animation under direct keyboard control. To end the display sequence hit the return key.
11. A(ddress) : this command selects the assumed address of the sprite data. This is used when the sprites are saved. If they are reloaded by a BASIC load command, that is where they will load.
12. S(ave) : asks for a file name, then saves the current range of sprites to disk, as a program file using the assumed address.
13. L(oad) : asks for a file name, then loads the current range of sprites from disk into the work area from that program file.
14. B(yte) : asks for a file name, then saves the current range of sprites to disk, as a sequential file compatible with Commodore's Assembler Development System.
15. Q(uit) : exits the program.
16. Z : rotates the current sprite. First, the center of rotation is requested on the bottom line of the information box. X values range from 0 to 23, while Y values range from 0 to 20. Then the angle of rotation is requested. Following normal conventions, clockwise angles are positive, while counter clockwise angles are negative. Note: often a sprite is distorted by rotation.
17. F1 : toggles the selection of the current sprite display area from under the creation box to under the information box and back again.
18. F3 : step through possible screen colors.

EDITING COMMAND KEYS

These commands operate on the current sprite in the creation box.

1. * : places a dot at the current cursor position.
2. SPACE BAR : places a space (removes a dot) from the current cursor position.
3. CRSR RIGHT : moves the cursor one position to the right.
4. CRSR LEFT : moves the cursor one position to the left.
5. CRSR UP : moves the cursor one line up.
6. CRSR DOWN : moves the cursor one line down.
7. RETURN : moves the cursor to the start of the current line.
8. HOME : moves the cursor to the top left corner of the sprite.
9. CLR : erases the current sprite.
10. RVS : reverses the current sprite.
11. INST : moves all dots on the current one place to the right.
12. DEL : moves all dots on the current line one place to the left.
13. + : moves all lines from the current line to the bottom line one line down.
14. - : moves all lines from the current line to the top line, one line up.

COLOR CODES

BLK	WHT	RED	CYN	PUR	GRN	BLU	YEL
ORN	BRN	RD2	GY1	GY2	GN1	BL2	GY3

CHARACTER EDITOR
(CHRED)

INTRODUCTION

This program allows editing of single characters, or editing four characters at a time (a 16 bit by 16 bit character). When editing expanded charactes, all character editor commands are adjusted to operate on the expanded character. More details on expansion can be found under the X(pand) command.

SCREEN FORMAT

The screen is basically divided into five areas by the character editor program. The first is the large character creation box. Here the character is formed by using the "*" and " " symbols. In this box, an entire character is represented.

The second area is the information box. Current character parameters are displayed here. Also, when a command needs a parameter to operate, a cursor will appear at the appropriate place in this box.

The third area is the strip below the creation box. Here is where the entire current character set is displayed. Also, the character currently being edited is highlighted in black.

The fourth area is the display area, where the current character appears.

The fifth area is the command menu, where most of the commands are displayed.

General Notes: The first letter of a command is enough. When a command requires an input (like character number), a cursor will appear in the information box. Type the answer there, ending it with a return. If you type an incorrect letter, use the delete key. If your response is illegal, the program will ignore it, keeping the old value of the parameter.

THE INDFORMATION BOX

The information box contains the following information:

	PARAMETER	POSSIBLE VALUES
1.	Character number	0-63
2.	Name of current file	any 5 letters
3.	Assumed address	any 4 digit hex number
4.	Range	0-63
5.	Type of character displayed	HIRES or MULTI
6.	Foreground color	any of the 16 colors
7.	Multi-color register 0	any of the 16 colors
8.	Multi-color register 1	any of the 16 colors
9.	X expand	YES or NO

10. The bottom blank line of the information box is used for other command inputs that don't need to be continually displayed.

COMMANDS

1. E(dit) : requests a character number from 0 to 63. That character becomes the current character. The current appearance of the character is displayed both in the creation box and the current display area.
2. N(ext character) : selects the next character as the current character.
3. T(ype) : selects between displaying the current character as a hires (h) or a multi-color (m) character.
4. M(ove here) : asks for a character number. That character is copied into the current character. Note that this operation destroys the current character.
5. C(olor) : allows choice of colors. A cursor will appear at each of the color parameters in turn. Type the three letter abbreviation for the color of your choice for each register in turn.
6. X(pand) : selects either 8 bit by 8 bit characters or 16 bit by 16 bit characters. Answer YES or NO (or Y or N) to select expansion. When expansion is selected, four contiguous characters will be treated as a single entity for the purpose of all commands. The creation box will be expanded. The characters are edited in the following format.

   1 3
   2 4
7. F(ont) : masking function from the Commodore 64 ROM. The program asks if you want to mask all the characters. Answering Y replaces the current character set with the upper case character set from ROM. If you answer N (or just hit RETURN), the program will ask for a character number (from 0 to 63). That character from ROM will be copied into the current character.
8. R(ange) : sets a range of characters for use with SAVE, LOAD, BYTE, and DISPLAY commands. The form is ##:## (i.e. 0:12), for first and last characters to be affected by an operation.
9. O(r) : like move, but doesn't erase the current character first.
10. A(ddress) : this command selects the assumed address of the character data. This is used when the characters are saved. If they are reloaded by a BASIC load command, that is where they will load.
11. S(ave) : asks for a file name, then saves the current range of characters to disk, as a program file using the assumed address.
12. L(oad) : asks for a file name, then loads the current range of characters from disk into the work area from that program file.
13. B(yte) : asks for a file name, then saves the current range of characters to disk, as a sequential file compatible with Commodore's Assembler Development System.
14. Q(uit) : exits the program.
15. V(alue) : displays the values of the bytes which make up the current character. These values appear next in the character display area. This display lasts until any key is hit. Then the normal character display reappears.
16. H(ex flag) : toggles whether the V(alue) display will be in decimal or in hex.
17. F3 : step through possible screen colors.

(CHRED) cont.

EDITING COMMAND KEYS

These commands operate on the current character in the creation box.

1. * : places a dot at the current cursor position.
2. SPACE BAR : places a space (removes a dot) from the current cursor position.
3. CRSR RIGHT : moves the cursor one position to the right.
4. CRSR LEFT : moves the cursor one position to the left.
5. CRSR UP : moves the cursor one line up.
6. CRSR DOWN : moves the cursor one line down.
7. RETURN : moves the cursor to the start of the current line.
8. HOME : moves the cursor to the top left corner of the character.
9. CLR : erases the current character.
10. RVS : reverses the current character.
11. INST : moves all dots on the current one place to the right.
12. DEL : moves all dots on the current line one place to the left.
13. + : moves all lines from the current line to the bottom line one line down.
14. - : moves all lines from the current line to the top line, one line up.

COLOR CODES

BLK	WHT	RED	CYN	PUR	GRN	BLU	YEL
ORN	BRN	RD2	GY1	GY2	GN1	BL2	GY3

25 Rows X [illegible] Columns

ADDR OFFSET		ROW \ COL	D	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	37	39
DEC	HEX	D	H	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27
000	0000	0	0																																								
040	0028	1	1																																								
080	0050	2	2																																								
120	0078	3	3																																								
160	00A0	4	4																																								
200	00C8	5	5																																								
240	00F0	6	6																																								
280	0118	7	7																																								
320	0140	8	8																																								
360	0168	9	9																																								
400	0190	10	A																																								
440	01B8	11	B																																								
480	01E0	12	C																																								
520	0208	13	D																																								
560	0230	14	E																																								
600	0258	15	F																																								
640	0280	16	10																																								
680	02A8	17	11																																								
720	02D0	18	12																																								
760	02F8	19	13																																								
800	0320	20	14																																								
840	0348	21	15																																								
880	0370	22	16																																								
920	0398	23	17																																								
960	03C0	24	18																																								

03E7

999

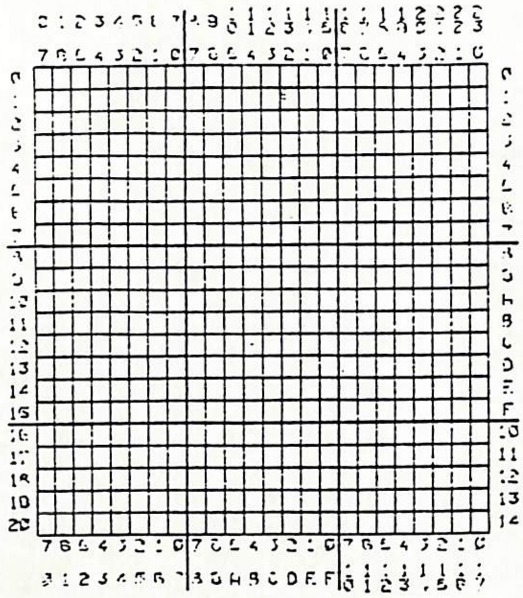
Video Matrix Reference

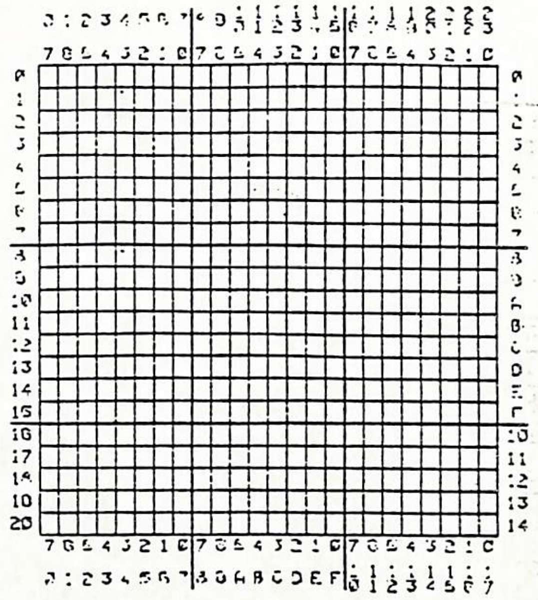
Chart/WORKSHEET

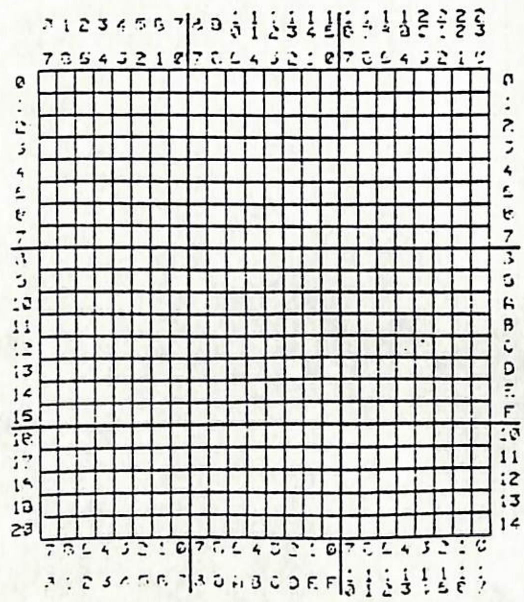
Notes:-

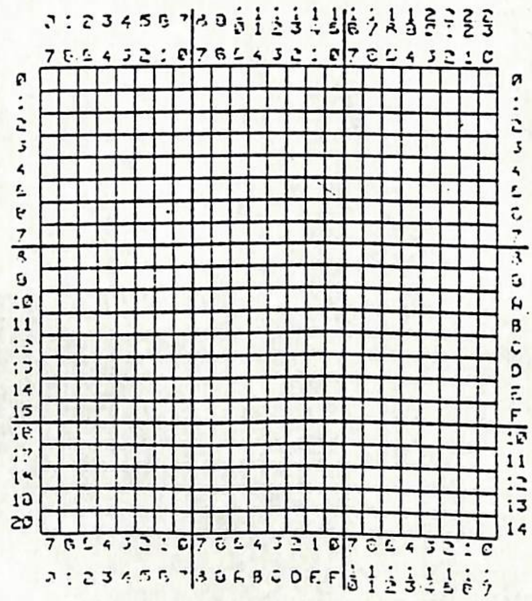
1) The base address in the Max System is $0400 (1024)
2) The sprite pointer address offset is $03F8 (thru $03FF)
3) The base address of the Color Matrix is $D800

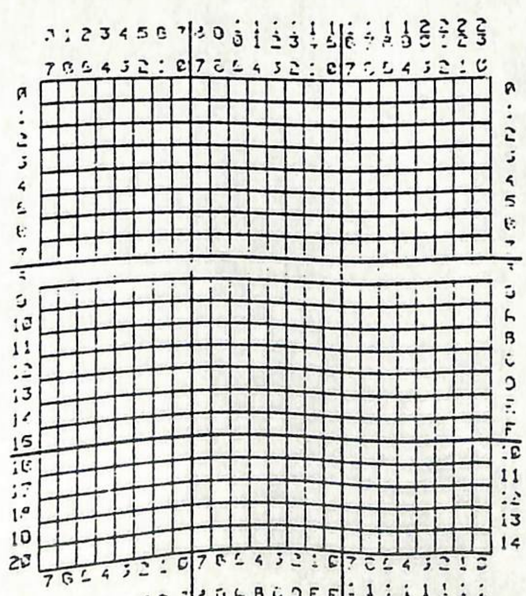
SPRITE WORKSHEET
(UNEXPANDED)

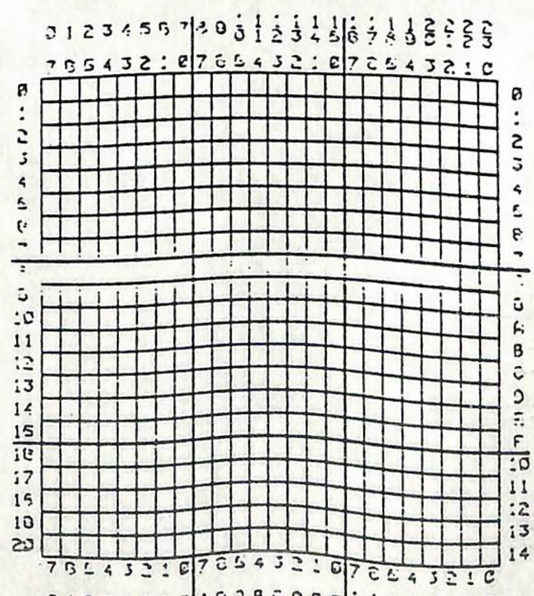




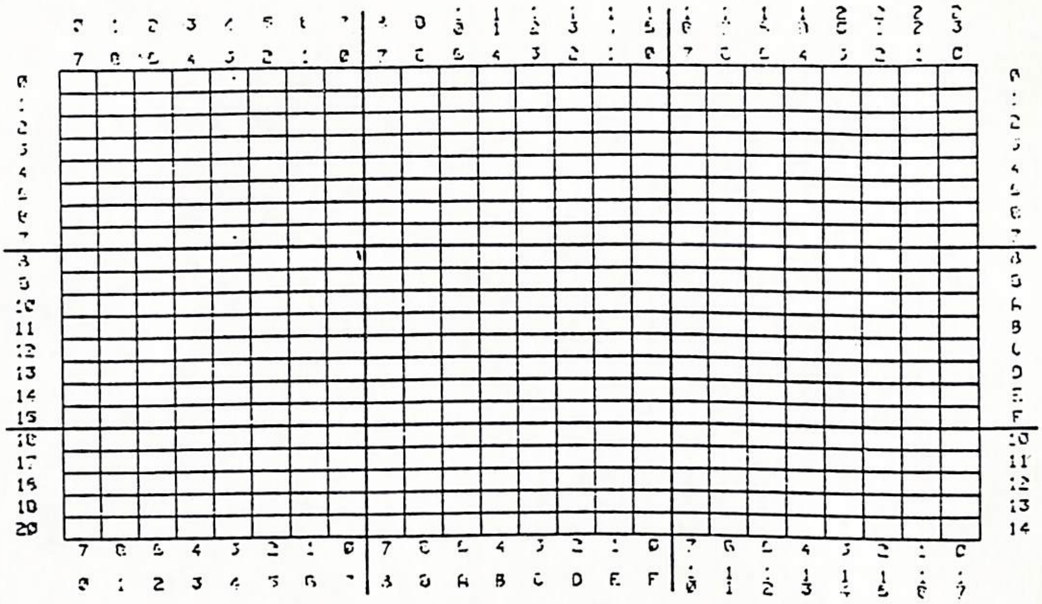


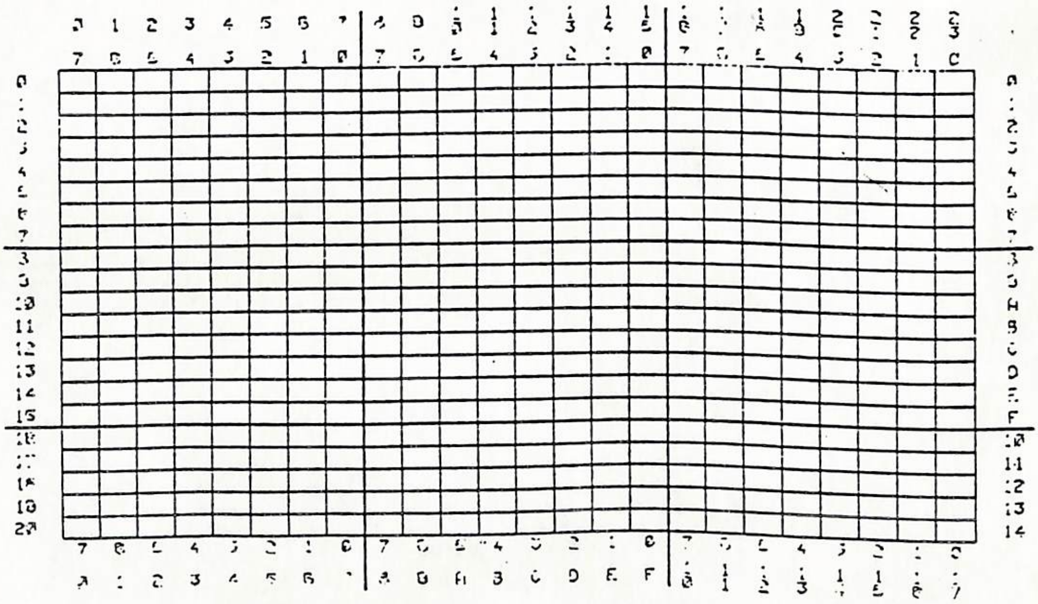


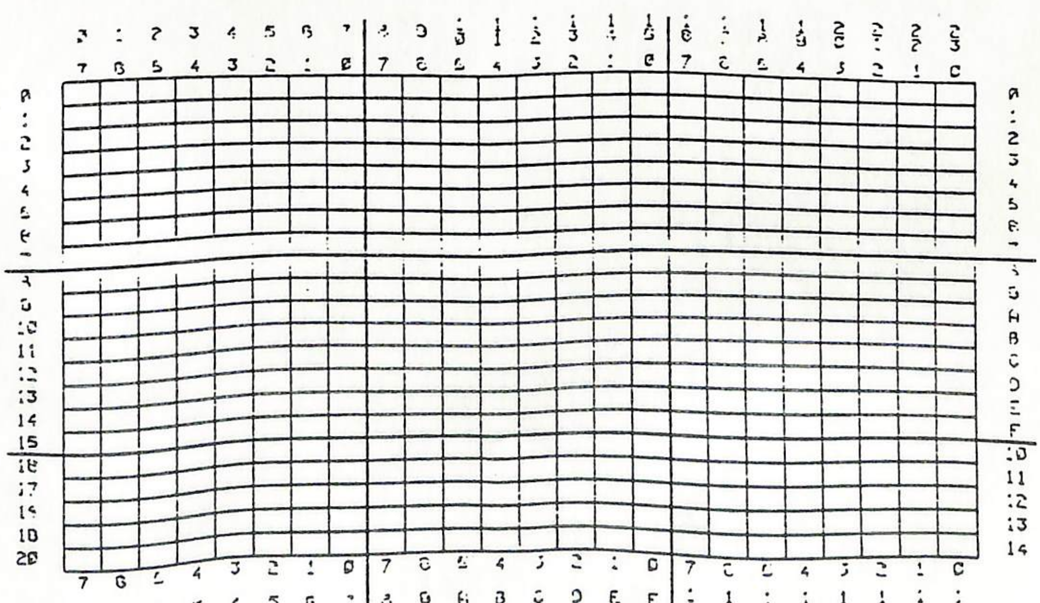




CEL SPRITE WORKSHEET
(X EXPANDED)

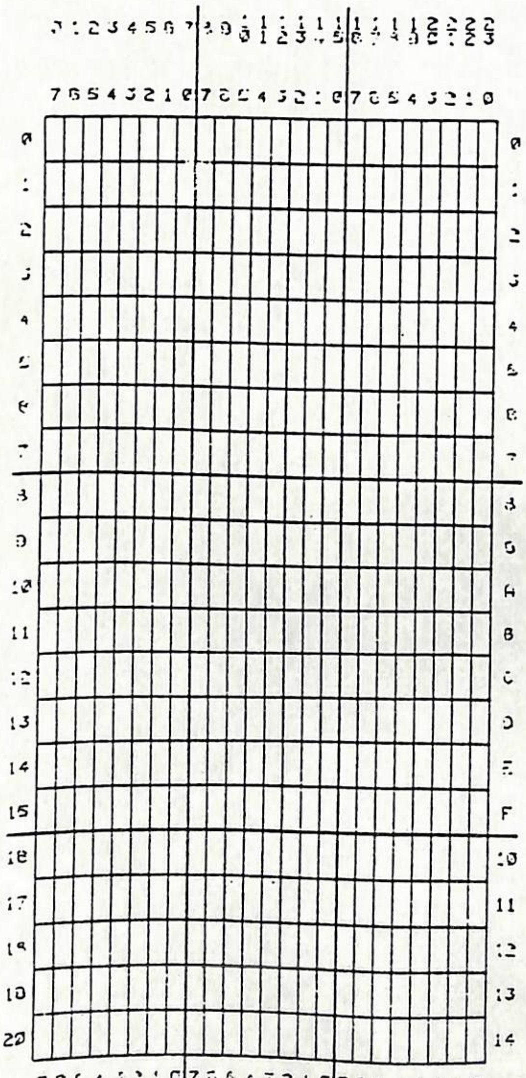


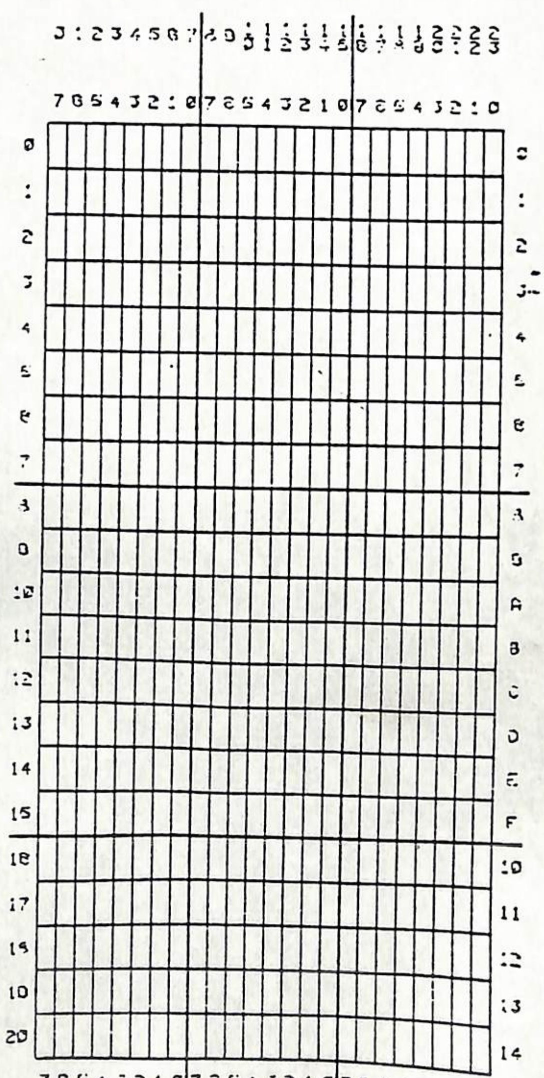




CEL SPRITE WORKSHEET

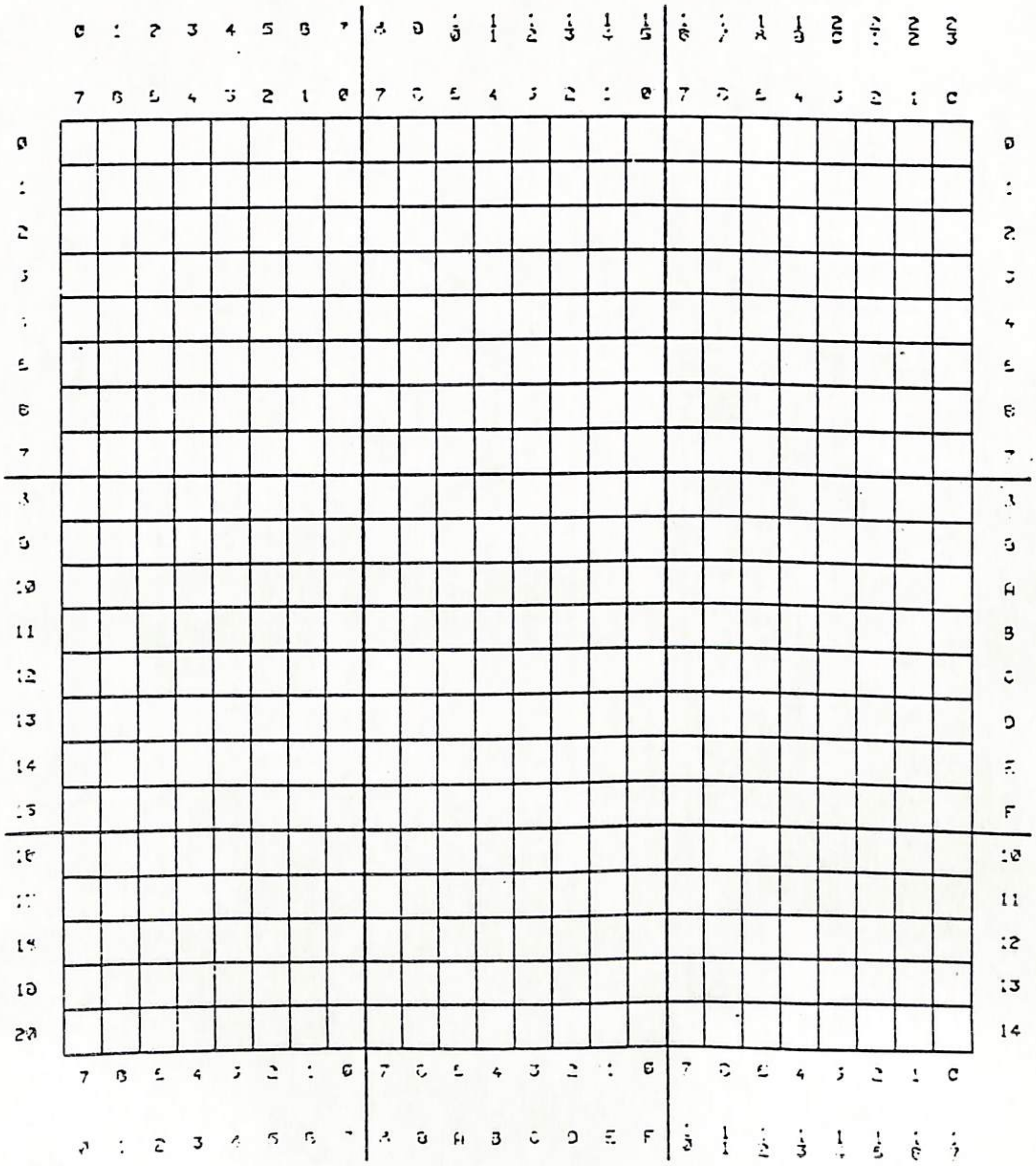
(Y EXPANDED)



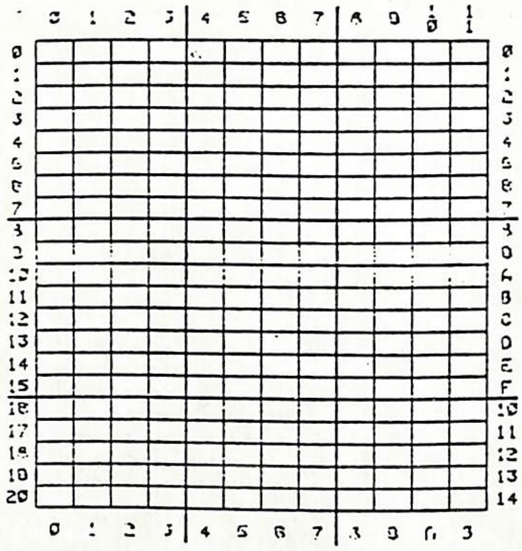


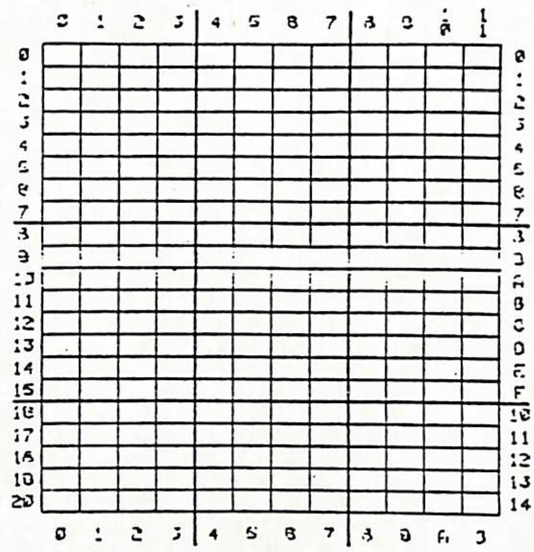
CEL SPRITE WORKSHEET

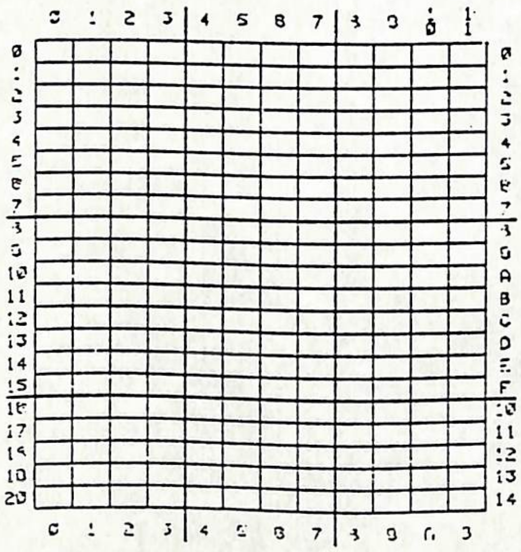
(X & Y EXPANDED)

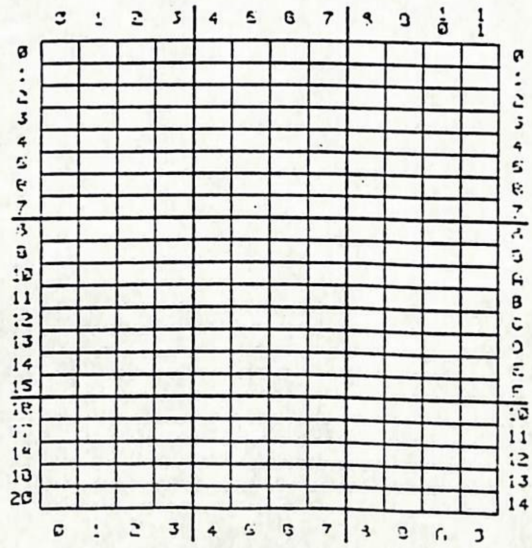


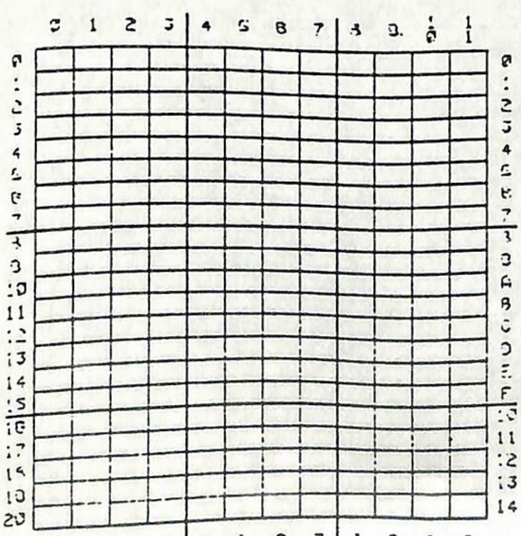
(UNEXPANDED)

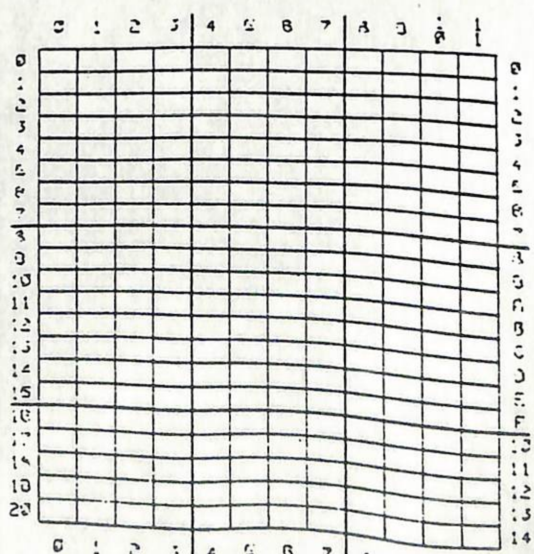




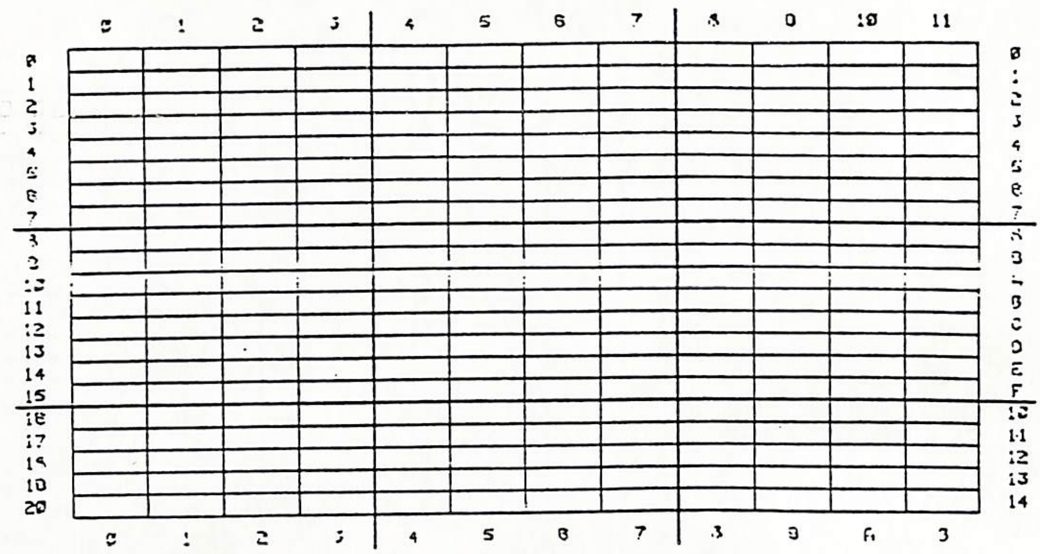


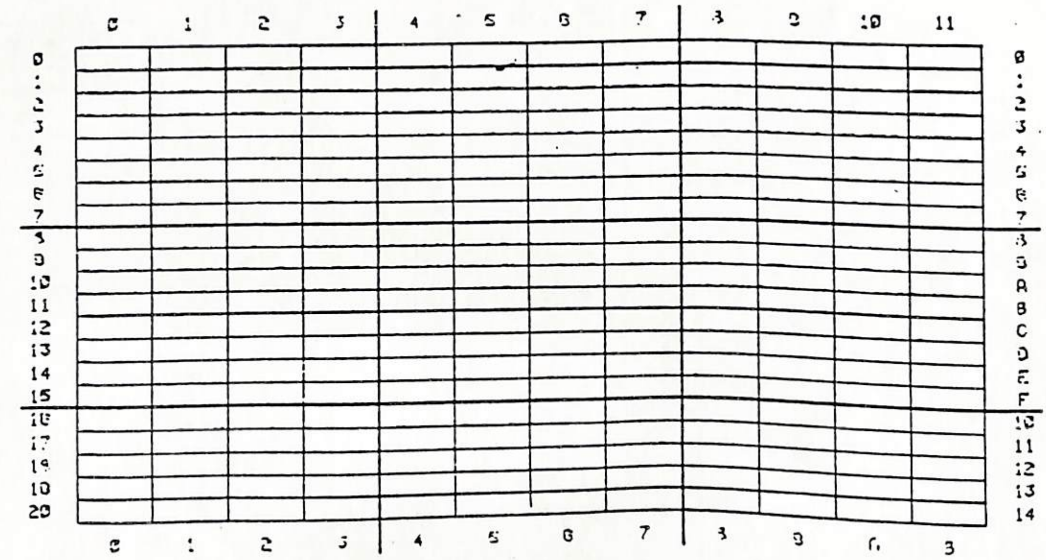


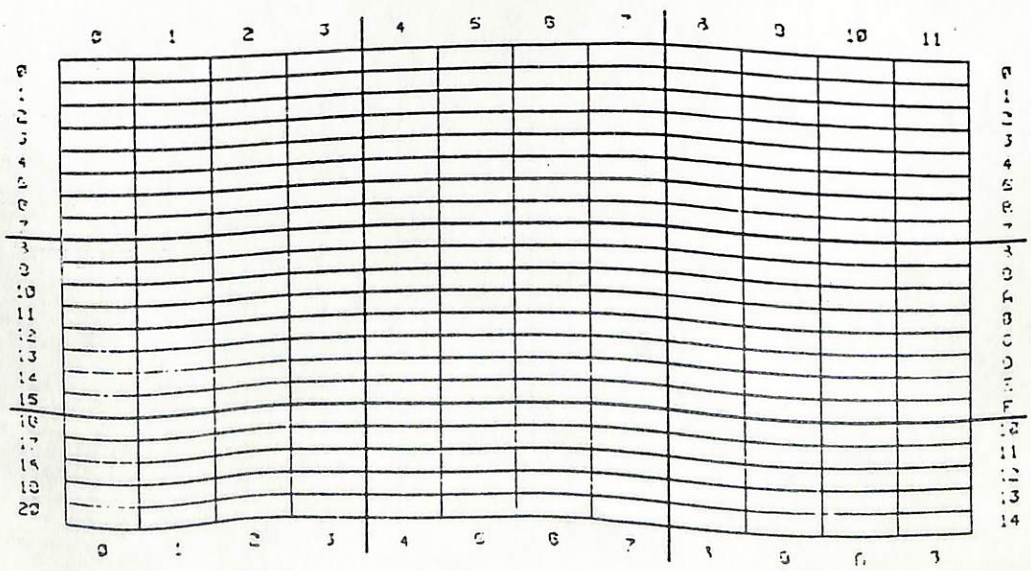




(X EXPANDED)

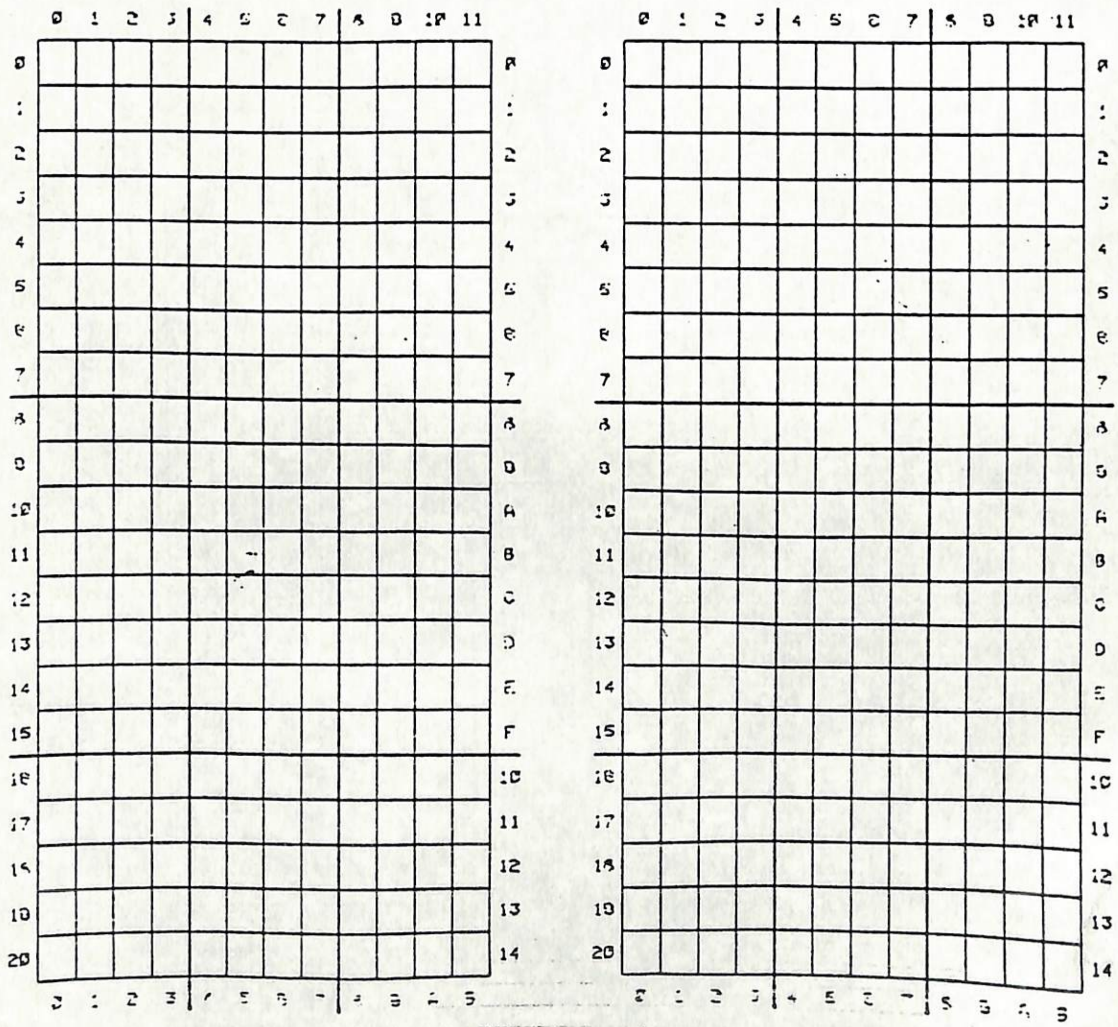






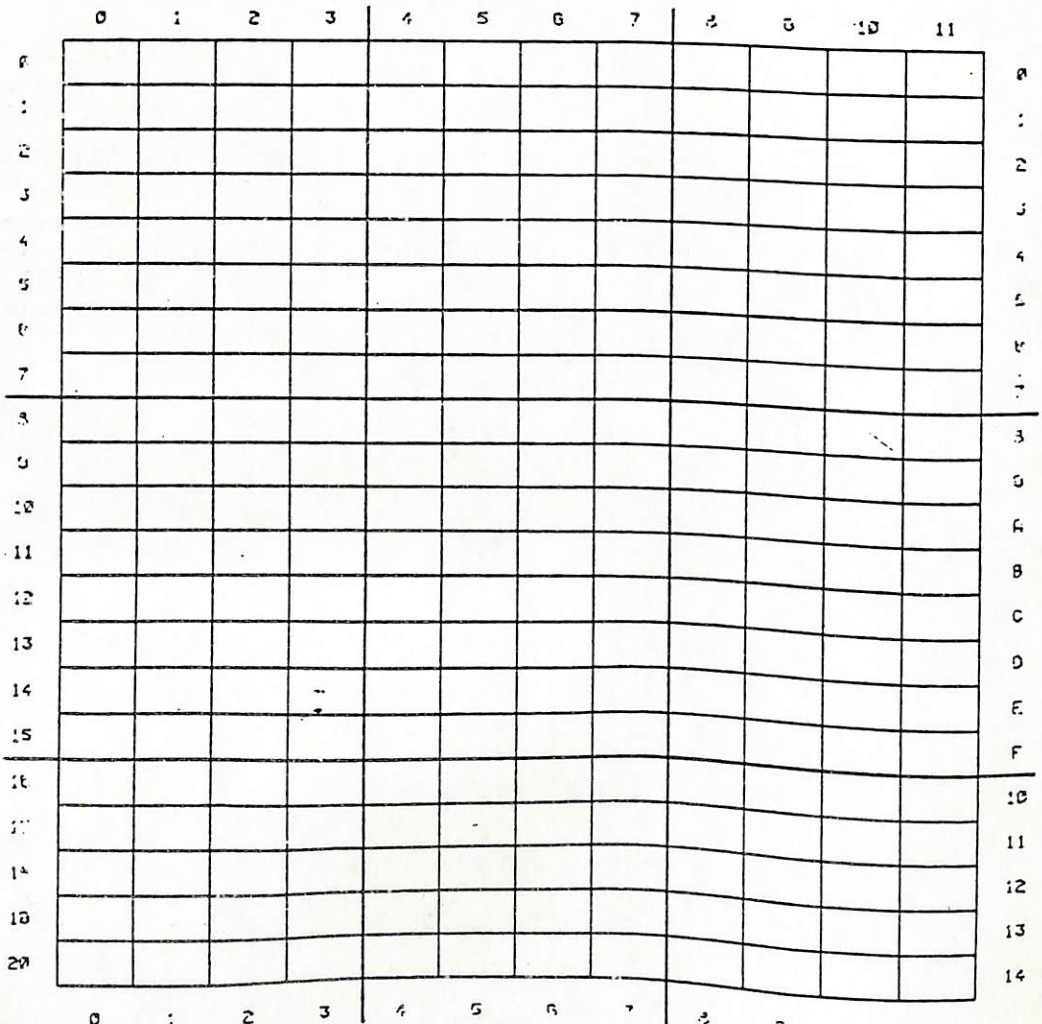
CEL MULTICOLOR SPRITE WORKSHEET

(Y EXPANDED)



CEL MULTICOLOR SPRITE WORKSHEET

(X & Y EXPANDED)



USERS GUIDE
FOR THE
6566/6567 VIDEO INTERFACE CHIPS

The 6566/6567 are multi-purpose color video controller devices for use in both computer video terminals and video game applications. Both devices contain 47 control registers which are accessed via a standard 8-bit microprocessor bus (65XX) and will access up to 16K of memory for display information. The various operating modes and options within each mode are described.

CHARACTER DISPLAY MODE

In the character display mode, the 6566/6567 fetches CHARACTER POINTERs from the VIDEO MATRIX area of memory and translates the pointers to character dot location addresses in the 2,048 byte CHARACTER BASE area of memory. The video matrix is comprised of 1,000 consecutive locations in memory which each contain an eight bit character pointer. The location of the video matrix within memory is defined by VM13-VM10 in register 24 ($18) which are used as the 4 MSB of the video matrix address. The lower order 10 bits are provided by an internal counter (VC9-VC0) which steps through the 1000 character locations. Note that the 6566/6567 provides 14 address outputs; therefore additional system hardware may be required for complete system memory decodes.

CHARACTER POINTER ADDRESS

A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00
VM13	VM12	VM11	VM10	VC9	VC8	VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0

The eight bit character pointer permits up to 256 different character definitions to be available simultaneously. Each character is an 8x8 dot matrix stored in the character base as eight consecutive bytes. The location of the character base is defined by CB13-CB11 also in register 24 ($18) which are used for the 3 most significant bits (MSB) of the character base address. The 11 lower order addresses are formed by the 8 bit character pointer from the video matrix (D7-D0) which selects a particiluar character, and a 3 bit raster counter (RC2-RC0) which selects one of the eight character bytes. The resulting characters are formated as 25 rows of 40 characters each. In addition to the 8 bit character pointer, a 4-bit COLOR NYBBLE is associated with each video matrix location (the video matrix memory must be 12 bits wide) which defines one of sixteen colors for each character.

CHARACTER DATA ADDRESS

A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00
CB13	CB12	CB11	D7	D6	D5	D4	D3	D2	D1	D0	RC2	RC1	RC0

STANDARD CHARACTER MODE (MCM = BMM = ECM = 0)

In the standard character mode, the 8 sequential bytes from the character base are displayed directly on the 8 lines in each character region. A "0" bit causes the background #0 color (from register 33 ($21)) to be displayed while the color selected by the color nybble (foreground) is displayed for a "1" bit (see Color Code Table).

FUNCTION	CHARACTER BIT	COLOR DISPLAYED
Background	0	Background #0 color (register 33 ($21))
Foreground	1	Color selected by 4-bit color nybble

Therefore, each character has a unique color determined by the 4-bit color nybble (1 of 16) and all characters share the common background color.

MULTICOLOR CHARACTER MODE (MCM = 1, BMM = ECM = 0)

Multi-color mode provides additional color flexibility allowing up to four colors within each character but with reduced resolution. The multi-color mode is selected by setting the MCM bit in register 22 ($16) to "1", which causes the dot data stored in the character base to be interpreted in a different manner. If the MSB of the color nybble is a "0", the character will be displayed as described in standard character mode, allowing the two modes to be inter-mixed (however, only the lower order 8 colors are available). When the MSB of the color nybble is a "1" (if MCM.MSB(CM) = 1) the character bits are interpreted in the multi-color mode:

FUNCTION	CHARACTER BIT PAIR	COLOR DISPLAYED
Background	00	Background #0 Color (register 33 ($21))
Background	01	Background #1 Color (register 34 ($22))
Foreground	10	Background #2 Color (register 35 ($23))
Foreground	11	Color specified by 3 LSB of color nybble

Since two bits are required to specify one dot color, the character is now displayed as a 4 x 8 matrix with each dot twice the horizontal size as in standard mode. Note, however, that each character region can now contain 4 different colors, two as foreground and two as background (see MOB priority).

EXTENDED COLOR MODE (ECM = 1, BMM = MCM = 0)

The extended color mode allows the selection of individual background colors for each character region with the normal 8 x 8 character resolution. This mode is selected by setting the ECM bit of register 17 ($11) to "1". The character dot data is displayed as in the standard mode (foreground color determined by the color nybble is displayed for a "1" data bit), but the 2 MSB of the character pointer are used to select the background color for each character region as follows:

CHAR POINTER MS BIT PAIR	BACKGROUND COLOR DISPLAYED FOR 0 BIT
00	Background #0 color (register 33 ($21))
01	Background #1 color (register 34 ($22))
10	Background #2 color (register 35 ($23))
11	Background #3 color (register 36 ($24))

Since the two MSB of the character pointers are used for color information, only 64 different character definitions are available. The 6566/6567 will force CB10 and CB9 to "0" regardless of the original pointer values, so that only the first 64 character definitions will be accessed. With extended color mode each character has one of sixteen individually defined foreground colors and one of the four available background colors.

NOTE — Extended color mode and multi-color mode should not be enabled simultaneously.

BIT MAP MODE

In bit map mode, the 6566/6567 fetches data from memory in a different fashion, so that a one-to-one correspondence exists between each displayed dot and a memory bit. The bit map mode provides a screen resolution of 320H x 200V individually controled display dots. Bit map mode is selected by setting the BMM bit in register 17 ($11) to a "1". The VIDEO MATRIX is still accessed as in character mode, but the video matrix data is no longer interpreted as character pointers, but rather as color data. The VIDEO MATRIX COUNTER is then also used as an address to fetch the dot data for display from the 8,000 byte DISPLAY BASE. The display base address is formed as follows:

A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00
CB13	VC9	VC8	VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0	RC2	RC1	RC0

VCx denotes the video matrix counter outputs, RCx denotes the 3 bit raster line counter and CB13 is from register 24 ($18). The video matrix counter steps through the same 40 locations for eight raster lines, continuing to the next 40 locations every eighth line, while the raster counter increments once for each horizontal video line (raster line). This addressing results in each eight sequential memory

locations being formatted as an 8x8 dot block on the video display.

STANDARD BIT MAP MODE (BMM = 1, MCM = 0)

When standard bit map mode is in use, the color information is derived only from the data stored in the video matrix (the color nybble is disregarded). The 8 bits are divided into two 4-bit nybbles which allow two colors to be independentaly selected in each 8x8 dot block. When a bit in the display memory is a "0" the color of the output dot is set by the least significant (lower) nybble (LSN). Similarly, a display memory bit of "1" selects the output color determined by the MSN (upper nybble).

BIT	DISPLAY COLOR
0	Lower nybble of video matrix pointer
1	Upper nybble of video matrix pointer

MULTI-COLOR BIT MAP MODE (BMM = MCM = 1)

Multi-colored bit map mode is selected by setting the MCM bit in register 22 ($16) to a "1" in conjunction with the BMM bit. Multi-color mode uses the same memory access sequence as standard bit map mode, but interprets the dot data as follows:

BIT PAIR	DISPLAY COLOR
00	Background #0 color (register 33 ($21))
01	Upper nybble of video matrix pointer
10	Lower nybble of video matrix pointer
11	Video matrix color nybble

Note that the color nybble (DB11-DB8) IS used for the multi-color bit map mode. Again, as two bits are used to select one dot color, the horizontal dot size is doubled, resulting in a screen resolution of 160H x 200V. Utilizing multi-color bit map mode, three independently selected colors can be displayed in each 8 x 8 block in addition to the background color.

MOVABLE OBJECT BLOCKS

The movable object block (MOB) is a special type of character which can be displayed at any one position on the screen without the block constraints inherent in character and bit map mode. Up to 8 unique MOBs can be displayed simultaneously, each defined by 63 bytes in memory which are displayed as a 24x21 dot array (shown below). A number of special features make MOBs especailly suited for video graphics and game applicatons.

BYTE	BYTE	BYTE
00	01	02
03	04	05
.	.	.
.	.	.
.	.	.
57	58	59
60	61	62

MOB DISPLAY BLOCK

ENABLE

Each MOB can be selectively enabled for display by setting its corresponding enable bit (MnE) to "1" in register 21 ($15). If the MnE bit is "0", no MOB operations will occur involving the disabled MOB.

POSITION

Each MOB is positioned via its X and Y position register (see register map) with a resolution of 512 horizontal and 256 vertical positions. The position of a MOB is determined by the upper-left corner of the array. X locations 24 to 343 ($18-$157) and Y locations 50 to 249 ($32-$F9) are visible. Since not all available MOB positions are entirely visible on the screen, MOBs may be moved smoothly on and off the display screen .

COLOR

Each MOB has a seperate 4-bit register to determine the MOB color. The two MOB color modes are:

STANDARD MOB (MnMC = 0)

In the standard mode, a "0" bit of MOB data is allows any background data to show through (transparent) and a "1" bit is displayed as the MOB color determined by the corresponding MOB Color register.

MULTI-COLOR MOB (MnMC = 1)

Each MOB can be individually selected as a multi-color MOB via MnMC bits in the MOB Multi-color register 28 ($1C). When the MnMC bit is "1", the corresponding MOB is displayed in the multi-color mode. In the multi color mode, the MOB data is interpreted in pairs (similar to the other multi-color modes) as follows:

BIT PAIR	COLOR DISPLAYED
00	Transparent
01	MOB Multi-color #0 (register 37 ($25))
10	MOB Color (registers 39-46 ($27-$2E))
11	MOB Multi-color #1 (register 38 ($26))

Since two bits of data are required for each color, the resolution of the MOB is reduced to 12x21, with each horizontal dot expanded to twice standard size so that the overall MOB size does not change. Note that up to 3 colors can be displayed in each MOB (in addition to transparent) but that two of the colors are shared among all the MOBs in the multi-color mode.

MAGNIFICATION

Each MOB can be selectively expanded (2X) in both the horizontal and vertical directions. Two registers contain the control bits (MnXE,MnYE) for the magnification control:

RESIGTER	FUNCTION
29 ($1D)	Horizontal expand MnXE - "1"=expand; "0"=normal
23 ($17)	Vertical expand MnYE - "1"=expand; "0"=normal

When MOBs are expanded, no increase in resolution is realized. The same 24x21 array (12x21 if multi-colored) is displayed, but the overall MOB dimension is doubled in the desired direction (the smallest MOB dot may be up to 4X standard dot dimension if a MOB is both multi-colored and expanded).

PRIORITY

The priority of each MOB may be individually controled with respect to the other displayed information from character or bit map modes. The priority of each MOB is set by the corresponding bit (MnDP) of register 27 ($1B) as follows:

REG BIT	PRIORITY TO CHARACTER OR BIT MAP DATA
0	Non-transparent MOB data will be displayed (MOB in front)
1	Non-transparent MOB data will be displayed only instead of Bkgd #0 or muli-color bit pair 01 (MOB behind)

MnDP = 1	MNDP = 0
MOBn	Foreground
Foreground	MOBn
Background	Background

MOB - DISPLAY DATA PRIORITY

MOB data bits of "0" ("00" in multi-color mode) are transparent, always permitting any other information to be displayed.

The MOBs have a fixed priority with respect to each other, with MOB 0 having the highest priority and MOB 7 the lowest. When MOB data (except transparent data) of two MOBs are co-incident, the data from the lower number MOB will be displayed. MOB vs. MOB data is prioritized before priority resolution with character or bit map data.

COLLISION DETECTION

Two types of MOB collision (co-incidence) are detected. MOB to MOB collision and MOB to display data collision:

A collision between two MOBs occurs when non-transparent output data of two MOBs are co-incident. Co-incidence of MOB transparent areas will not generate a collision. When a collision occurs, the MOB bits (MnM) in the MOB-MOB COLLISION register 30 ($1E) will be set to "1" for both colliding MOBs. As a collision between two (or more) MOBs occurs, the MOB-MOB collision bit for each collided MOB will be set. The collision bits remain set until a read of the collision register, when all bits are automatically cleared. MOBs collisions are detected even if positioned off-screen.

The second type of collision is a MOB-DATA collision between a MOB and foreground display data from the character or bit map modes. The MOB-DATA COLLISION register 31 ($1F) has a bit (MnD) for each MOB which is set to "1" when both the MOB and non-background display data are co-incident. Again, the co-incidence of only transparent data does not generate a collision. For special applications, the display data from the 0.1 multicolor bit pair also does not cause a collision. This feature permits their use as background display data without interfering with true MOB collisions. A MOB-DATA collision can occur off-screen in the horizontal direction if actual display data has been scrolled to an off-screen position (see scrolling). The MOB-DATA COLLISION register also automatically clears when read.

The collision interrupt latches are set whenever the first bit of either register is set to "1". Once any collision bit within a register is set high, subsequent collisions will not set the interrupt latch until that collision register has been cleared to all "0"s by a read.

MOB MEMORY ACCESS

The data for each MOB is stored in 63 consecutive bytes of memory. Each block of MOB data is defined by a MOB pointer, located at the end of the VIDEO MATRIX. Only 1,000 bytes of the video matrix are used in the normal display modes, allowing the video matrix locations 1016-1023 (VM base+$3F8 to VM base+$3FF) to be used for MOB pointers 0-7, respectively. The eight-bit MOB pointer from the video matrix together with the six bits from the MOB byte counter (to address 63 bytes) define the entire 14-bit address field:

A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00
MP7	MP6	MP5	MP4	MP3	MP2	MP1	MP0	MC5	MC4	MC3	MC2	MC1	MC0

Where MPx are the MOB pointer bits from the video matrix and MCx are the internally generated MOB counter bits. The MOB pointers are read from the video matrix at the end of every raster line. When the Y position register of a MOB matches the current raster line count, the actual fetches of MOB data begin. Internal counters automatically step through the 63 bytes of MOB data, displaying three bytes on each raster line.

OTHER FEATURES

SCREEN BLANKING

The display screen may be blanked by setting the DEN bit in register 17 ($11) to a "0". When the screen in blanked, the entire screen will be filled with the exterior color as set in register 32 ($20). When blanking is active, only transparent (Phase 1) memory accesses are required, permitting full processor utilization of the system bus. MOB data, however, will be accessed if the MOBS are not also disabled. The DEN bit must be set to "1" for normal video display.

ROW/COLUMN SELECT

The normal display consists of 25 rows of 40 characters (or character regions) per row. For special display purposes, the display window may be reduced to 24 rows and 38 characters. There is no change in the format of the displayed information, except that characters (bits) adjacent to the exterior border area will now be covered by the border. The select bits operate as follows:

RSEL	Number of rows	CSEL	Number of columns
0	24 rows	0	38 columns
1	25 rows	1	40 columns

The RSEL bit is in register 17 ($11) and the CSEL bit is in register 22 ($16). For standard display the larger display window is normally used, while the smaller display window is normally used in conjunction with scrolling.

SCROLLING

The display data may be scrolled up to one entire character space in both the horizontal and vertical direction. When used in conjunction with the smaller display window (above), scrolling can be used to create a smooth panning motion of display data while updating the system memory only when a new character row (or column) is required. Scrolling is also used to center a fixed display within the display window.

BITS	REGISTER	FUNCTION
X2,X1,X0	22 ($16)	Horizontal Position
Y2,Y1,Y0	17 ($11)	Vertical Position

LIGHT PEN

The light pen input latches the current screen position into a pair of registers (LPX,LPY) on a low-going edge. The X position register 19 ($13) will contain the 8 MSB of the X position at the time of transition.

The Y position is latched in its register 20 ($14) but here 8 bits provide single raster resolution within the visible display. The light pen latch may be triggered only once per frame, and subsequent triggers within the same frame will have no effect.

RASTER REGISTER

The raster register is a dual function register. A read of the raster register 18 ($12) returns the lower 8 bits of the current raster positon (the MSB-RC8 is located in register 17 ($11)). The raster register can be interrogated to implement display changes outside the visible area to prevent display flicker. The visible display window is from raster 50 through raster 249 ($033-$0F8). A write to the raster bits (including RC8) is latched for use in an internal raster compare. When the current raster matches the written value, the raster interrupt latch is set.

INTERRUPT REGISTER

The interrupt register shows the status of the four sources of interrupt. An interrupt latch in register 25 ($19) is set to "1" when an interrupt source has generated an interrupt request. The four sources of interrupt are:

LATCH BIT	ENABLE BIT	WHEN SET
IRST	ERST	Set when (raster count) = (stored raster count)
IMDC	EMDC	Set by MOB-DATA collision register (first collision only)
IMMC	EMMC	Set by MOB-MOB collision register (first collision only)
ILP	ELP	Set by negative transition of LP input (once per frame)
IRQ		Set high by latch set and enabled (invert of IRQ/ output)

To enable an interrupt request to set the IRQ/ output to "0", the corresponding interrupt enable bit in register 26 ($1A) must be set to "1". Once an interrupt latch has been set, the latch CAN ONLY BE cleared only by writing a "1" to the desired latch in the interrupt register. This feature allows selective handling of video interrupts without software required to "remember" active interrupts.

DYNAMIC RAM REFRESH

A dynamic ram refresh controller is built in to the 6566/6567 devices. Five 8-bit row addresses are refreshed every raster line. This rate guarantees a maximum delay of 2.02 ms between the refresh of any single row address in a 128 refresh scheme. (The maximum delay is 3.56ms in a 256 address refresh scheme). This refresh is totally transparent to the system, since the refresh occurs during Phase 1 of the system clock. The 6567 generates both RAS/ and CAS/ which are normally connected directly to the dydnamic rams. RAS/ and CAS/ are generated for every Phase 2 and every video data access (including refresh) so that external clock generation is not required.

RESET

The reset bit (RES) in register 22 ($16) is not used for normal operation. Therefore it should be set to "0" when initializing the video chip. When set to a "1", the entire operation of the video chip is suspended, including video outputs and sync, memory refresh, and system bus access.

THEORY OF OPERATION

SYSTEM INTERFACE

The 6566/6567 video controller devices interact with the system data bus in a special way. A 65XX system requires requires the system busses only during the Phase 2 (clock high) portion of the cycle. The 6566/6567 devices take advantage of this feature by normally accessing system memory during the Phase 1 (clock low) portion of the clock cycle. Therefore, operations such as character data fetches and memory refresh are totally transparent to the processor and do not reduce the processor through-put. The video chips provide the interface control signals required to maintain this bus sharing.

The video devices provide the signal AEC (address enable control) which is used to disable the processor address bus drivers allowing the video device to access the address bus. AEC is active low which permits direct connection to the AEC input of the 65XX family. The AEC signal is normally activated during Phase 1 so that processor operation is not affected. Because of this bus "sharing", all memory accesses must be completed in 1/2 cycle. Since the video chips provide a 1Mhz clock (which must be used as system Phase 2), a memory cycle is 500ns including address setup, data access and data setup to the reading device.

Certain operations of the 6566/6567 require data at a faster rate than available by reading only during the Phase 1 time; specifically, the access of character pointers from the video matrix and the fetch of MOB data. Therefore, the processor must be disabled and the data accessed during the Phase 2 clock. This is accomplished via the BA (bus available) signal. The BA line is normally high but is brought low during Phase 1 to indicate that the video chip will require a Phase 2 data access. Three Phase 2 times are allowed after BA low for the processor to complete any current memory accesses. On the fourth Phase 2 after BA low, the AEC signal will remain low during Phase 2 as the video chip fetches data. The BA line is normally connected to the RDY input of a 65XX processor. The character pointer fetches occur every eighth raster line during the display window and require 40 consecutive Phase 2 accesses to fetch the video matrix pointers. The MOB data fetches require 4 memory accesses as follows:

PHASE	DATA	CONDITION
1	MOB Pointer	Every raster
2	MOB Byte 1	Each raster while MOB is displayed
1	MOB Byte 2	Each raster while MOB is displayed
2	MOB Bbyte3	Each raster while MOB is displayed

The MOB pointers are fetched every other Phase 1 at the end of each raster line. As required, the additional cycles are used for MOB data fetches. Again, all necessary bus control is provided by the 6566/6567 devices.

MEMORY INTERFACE

The two versions of the video interface chip, 6566 and 6567, differ in address output configurations. The 6566 has thirteen fully decoded addresses for direct connection to the system address buss. The 6567 has multiplexed addresses for direct connection to 64K dynamic rams. The least significant address bits, A06-A00, are present on A06-A00 while RAS/ is brought low, while the most significant bits, A13-A08, are present on A05-A00 while CAS/ is brought low. The pins A11-A07 on the 6567 are static address outputs to allow direct connection of these bits to a conventional 16K (2Kx8) ROM. (The lower order addresses require external latching).

PROCESSOR INTERFACE

Aside from the special memory accesses described above, the 6566/6567 registers can be accessed similar to any other peripheral device. The following processor interface signals are provided:

DATA BUS (DB7-DB0)

The eight data bus pins are the bi-directional data port, controlled by CS/, RW, and Phase 0. The data bus can only be accessed while AEC and Phase 0 are high and CS/ is low.

CHIP SELECT (CS/)

The chip select pin, CS/, is brought low to enable access to the device registers in conjunction with the address and RW pins. CS/ low is recoginzed only while AEC and Phase 0 are high.

READ/WRITE (R/W)

The read/write input, R/W, is used to determine the direction of data transfer on the data bus, in conjunction with CS/. When R/W is high ("1") data is transfered from the selected register to the data bus output. When R/W is low ("0") data presented on the data bus pins is loaded into the selected register.

ADDRESS BUS (A05-A00)

The lower six address pins, A5-A0, are bi-directional. During a processor read or write of the video device, these address pins are inputs. The data on the address inputs selects the register for read or write as defined in the register map.

CLOCK OUT (PH0)

The clock output, Phase 0, is the 1Mhz clock used as the 65XX processor Phase 0 in. All system bus activity is referenced to this clock. The clock frequency is generated by dividing the 8Mhz video input clock by eight.

INTERRPUTS (IRQ/)

The interrupt output, IRQ/, is brought low when an enabled source of interrupt occurs with in the device. The IRQ/ output is open drain, requiring an external pull-up resistor.

VIDEO INTERFACE

The video output signal from the 6566/6567 consists of two signals which must be externally mixed together. SYNC/LUM output contains all the video data, including horizontal and vertical syncs, as well as the luminance information of the video display. SYNC/LUM is open drain, requiring an external pullup of 500 ohms. The COLOR output contains all the chrominance information, including the color reference burst and the color of all display data. The COLOR output is open source and sould be terminated with 1,000 ohms to ground. After appropriate mixing of these two signals, the resulting signal can directly drive a video monitor or be fed to a modulator for use with a standard television.

AEC	PH0	CS/	R/W	ACTION
0	0	X	X	PHASE 1 FETCH, REFRESH
0	1	X	X	PHASE 2 FETCH (PROCESSOR OFF)
1	0	X	X	NO ACTION
1	1	0	0	WRITE TO SELECTED REGISTER
1	1	0	1	READ FROM SELECTED REGISTER
1	1	1	X	NO ACTION

SUMMARY OF 6566/6567 BUS ACTIVITY

```
        DB6    -|01            40|-  VCC
        DB5    -|02            39|-  DB7
        DB4    -|03            38|-  DB8
        DB3    -|04            37|-  DB9
        DB2    -|05            36|-  DB10
        DB1    -|06            35|-  DB11
        DB0    -|07            34|-  A10
        IRQ/   -|08            33|-  A09
        LP     -|09            32|-  A08
        CS/    -|10            31|-  A07
        R/W    -|11            30|-  A06      ("1")
        BA     -|12            29|-  A05      (A13)
        VDD    -|13            28|-  A04      (A12)
        COLOR  -|14            27|-  A03      (A11)
        S/LUM  -|15            26|-  A02      (A10)
        AEC    -|16            25|-  A01      (A09)
        PH0    -|17            24|-  A00      (A08)
        RAS/   -|18            23|-  A11
        CAS/   -|19            22|-  PHIN
        VSS    -|20            21|-  PHCL
```

6567 PINOUT DIAGRAM

(Multiplexed addresses in parentheses)

```
        DB6    -|01            40|-  VCC
        DB5    -|02            39|-  DB7
        DB4    -|03            38|-  DB8
        DB3    -|04            37|-  DB9
        DB2    -|05            36|-  DB10
        DB1    -|06            35|-  DB11
        DB0    -|07            34|-  A13
        IRQ/   -|08            33|-  A12
        LP     -|09            32|-  A11
        CS/    -|10            31|-  A10
        R/W    -|11            30|-  A09
        BA     -|12            29|-  A08
        VDD    -|13            28|-  A07
        COLOR  -|14            27|-  A06
        S/LUM  -|15            26|-  A05
        AEC    -|16            25|-  A04
        PH0    -|17            24|-  A03
        PHIN   -|18            23|-  A02
        PHCOL  -|19            22|-  A01
        VSS    -|20            21|-  A00
```

6566 PINOUT DIAGRAM

REGISTER MAP

ADDRESS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	DESCRIPTION
00 ($00)	M0X7	M0X6	M0X5	M0X4	M0X3	M0X2	M0X1	M0X0	MOB 0 X-position
01 ($01)	M0Y7	M0Y6	M0Y5	M0Y4	M0Y3	M0Y2	M0Y1	M0Y0	MOB 0 Y-position
02 ($02)	M1X7	M1X6	M1X5	M1X4	M1X3	M1X2	M1X1	M1X0	MOB 1 X-position
03 ($03)	M1Y7	M1Y6	M1Y5	M1Y4	M1Y3	M1Y2	M1Y1	M1Y0	MOB 1 Y-position
04 ($04)	M2X7	M2X6	M2X5	M2X4	M2X3	M2X2	M2X1	M2X0	MOB 2 X-position
05 ($05)	M2Y7	M2Y6	M2Y5	M2Y4	M2Y3	M2Y2	M2Y1	M2Y0	MOB 2 Y-position
06 ($06)	M3X7	M3X6	M3X5	M3X4	M3X3	M3X2	M3X1	M3X0	MOB 3 X-position
07 ($07)	M3Y7	M3Y6	M3Y5	M3Y4	M3Y3	M3Y2	M3Y1	M3Y0	MOB 3 Y-position
08 ($08)	M4X7	M4X6	M4X5	M4X4	M4X3	M4X2	M4X1	M4X0	MOB 4 X-position
09 ($09)	M4Y7	M4Y6	M4Y5	M4Y4	M4Y3	M4Y2	M4Y1	M4Y0	MOB 4 Y-position
10 ($0A)	M5X7	M5X6	M5X5	M5X4	M5X3	M5X2	M5X1	M5X0	MOB 5 X-position
11 ($0B)	M5Y7	M5Y6	M5Y5	M5Y4	M5Y3	M5Y2	M5Y1	M5Y0	MOB 5 Y-position
12 ($0C)	M6X7	M6X6	M6X5	M6X4	M6X3	M6X2	M6X1	M6X0	MOB 6 X-position
13 ($0D)	M6Y7	M6Y6	M6Y5	M6Y4	M6Y3	M6Y2	M6Y1	M6Y0	MOB 6 Y-position
14 ($0E)	M7X7	M7X6	M7X5	M7X4	M7X3	M7X2	M7X1	M7X0	MOB 7 X-position
15 ($0F)	M7Y7	M7Y6	M7Y5	M7Y4	M7Y3	M7Y2	M7Y1	M7Y0	MOB 7 Y-position
16 ($10)	M7X8	M6X8	M5X8	M4X8	M3X8	M2X8	M1X8	M0X8	MSB of X-position
17 ($11)	RC8	ECM	BMM	DEN	RSEL	Y2	Y1	Y0	See text
18 ($12)	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	Raster register
19 ($13)	LPX8	LPX7	LPX6	LPX5	LPX4	LPX3	LPX2	LPX1	Light Pen X
20 ($14)	LPY7	LPY6	LPY5	LPY4	LPY3	LPY2	LPY1	LPY0	Light Pen Y
21 ($15)	M7E	M6E	M5E	M4E	M3E	M2E	M1E	M0E	MOB Enable
22 ($16)	-	-	RES	MCM	CSEL	X2	X1	X0	See text
23 ($17)	M7YE	M6YE	M5YE	M4YE	M3YE	M2YE	M1YE	M0YE	MOB Y-expand
24 ($18)	VM13	VM12	VM11	VM10	CB13	CB12	CB11	-	Memory Pointers
25 ($19)	IRQ	-	-	-	ILP	IMMC	IMBC	IRST	Interrupt Register
26 ($1A)	-	-	-	-	ELP	EMMC	EMBC	ERST	Enable Interrupt
27 ($1B)	M7DP	M6DP	M5DP	M4DP	M3DP	M2DP	M1DP	M0DP	MOB-DATA Priority
28 ($1C)	M7MC	M6MC	M5MC	M4MC	M3MC	M2MC	M1MC	M0MC	MOB Multicolor Sel
29 ($1D)	M7XE	M6XE	M5XE	M4XE	M3XE	M2XE	M1XE	M0XE	MOB X-expand
30 ($1E)	M7M	M6M	M5M	M4M	M3M	M2M	M1M	M0M	MOB-MOB Collision
31 ($1F)	M7D	M6D	M5D	M4D	M3D	M2D	M1D	M0D	MOB-DATA Collision
32 ($20)	-	-	-	-	EC3	EC2	EC1	EC0	Exterior Color
33 ($21)	-	-	-	-	B0C3	B0C2	B0C1	B0C0	Bkgd #0 Color
34 ($22)	-	-	-	-	B1C3	B1C2	B1C1	B1C0	Bkgd #1 Color
35 ($23)	-	-	-	-	B2C3	B2C2	B2C1	B2C0	Bkgd #2 Color
36 ($24)	-	-	-	-	B3C3	B3C2	B3C1	B3C0	Bkgd #3 Color
37 ($25)	-	-	-	-	MM03	MM02	MM01	MM00	MOB Multicolor #0
38 ($26)	-	-	-	-	MM13	MM12	MM11	MM10	MOB Multicolor #1
39 ($27)	-	-	-	-	M0C3	M0C2	M0C1	M0C0	MOB 0 Color
40 ($28)	-	-	-	-	M1C3	M1C2	M1C1	M1C0	MOB 1 Color
41 ($29)	-	-	-	-	M2C3	M2C2	M2C1	M2C0	MOB 2 Color
42 ($2A)	-	-	-	-	M3C3	M3C2	M3C1	M3C0	MOB 3 Color
43 ($2B)	-	-	-	-	M4C3	M4C2	M4C1	M4C0	MOB 4 Color
44 ($2C)	-	-	-	-	M5C3	M5C2	M5C1	M5C0	MOB 5 Color
45 ($2D)	-	-	-	-	M6C3	M6C2	M6C1	M6C0	MOB 6 Color
46 ($2E)	-	-	-	-	M7C3	M7C2	M7C1	M7C0	MOB 7 Color

NOTE: A dash indicates a no connect. All no connects are read as a "1".

COLOR CODES

D3	D2	D1	D0	HEX	COLOR
0	0	0	0	0	BLACK
0	0	0	1	1	WHITE
0	0	1	0	2	RED
0	0	1	1	3	CYAN
0	1	0	0	4	PURPLE
0	1	0	1	5	GREEN
0	1	1	0	6	BLUE
0	1	1	1	7	YELLOW
1	0	0	0	8	ORANGE
1	0	0	1	9	BROWN
1	0	1	0	A	LT RED
1	0	1	1	B	DARK GREY
1	1	0	0	C	MED GREY
1	1	0	1	D	LT GREEN
1	1	1	0	E	LT BLUE
1	1	1	1	F	LT GREY

SECTION II

6581 Sound Interface Device (SID)

commodore
mos technology
NMOS
950 Rittenhouse Rd., Norristown, PA 19403 • Tel: 215/666-7950 • TWX: 510-660-4168

6581 SOUND INTERFACE DEVICE <SID>

CONCEPT

The 6581 Sound Interface Device (SID) is a single-chip, 3-voice electronic music synthesizer/sound effects generator compatible with the 65XX and similar microprocessor families. SID provides wide-range, high-resolution control of pitch (frequency), tone color (harmonic content) and dynamics (volume). Specialized control circuitry minimizes software overhead, facilitating use in arcade/home video games and low-cost musical instruments.

FEATURES

- 3 TONE OSCILLATORS
  - Range: 0-4 kHz
- 4 WAVEFORMS PER OSCILLATOR
  - Triangle, Sawtooth,
  - Variable Pulse, Noise
- 3 AMPLITUDE MODULATORS
  - Range: 48 dB
- 3 ENVELOPE GENERATORS
  - Exponential response
  - Attack Rate: 2mS-8S
  - Decay Rate: 6mS-24S
  - Sustain Level: 0-peak volume
  - Release Rate: 6mS-24S
- OSCILLATOR SYNCHRONIZATION
- RING MODULATION
- PROGRAMMABLE FILTER
  - Cutoff range: 30 Hz-12 kHz
  - 12 dB/octave Rolloff
  - Low pass, Band pass,
  - High pass, Notch outputs
  - Variable Resonance
- MASTER VOLUME CONTROL
- 2 A/D POT INTERFACES
- RANDOM NUMBER/MODULATION GENERATOR
- EXTERNAL AUDIO INPUT

6581 PIN CONFIGURATION

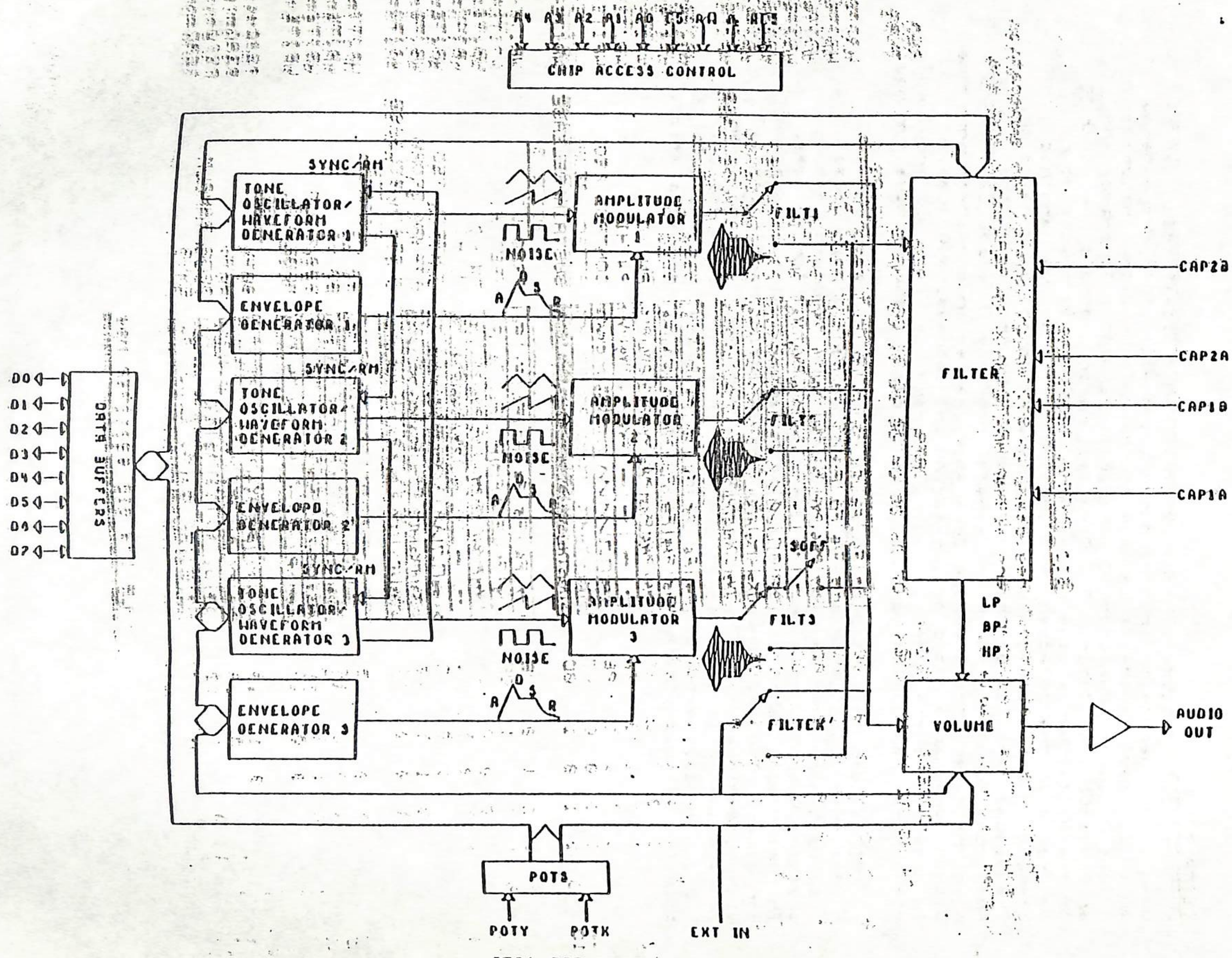
6581 SID

Signal	Pin	Pin	Signal
CAP1A	1	28	Vdd
CAP1B	2	27	AUDIO OUT
CAP2A	3	26	EXT IN
CAP2B	4	25	Vcc
$\overline{RES}$	5	24	POT X
Ø2	6	23	POT Y
R/$\overline{W}$	7	22	D7
$\overline{CS}$	8	21	D6
A0	9	20	D5
A1	10	19	D4
A2	11	18	D3
A3	12	17	D2
A4	13	16	D1
GND	14	15	D0

DESCRIPTION

The 6581 consists of three synthesizer "voices" which can be used independently or in conjunction with each other (or external audio sources) to create complex sounds. Each voice consists of a Tone Oscillator/Waveform Generator, an Envelope Generator and an Amplitude Modulator. The Tone Oscillator controls the pitch of the voice over a wide range. The Oscillator produces four waveforms at the selected frequency, with the unique harmonic content of each waveform providing simple control of tone color. The volume dynamics of the oscillator are controlled by the Amplitude Modulator under the direction of the Envelope Generator. When triggered, the Envelope Generator creates an amplitude envelope with programmable rates of increasing and decreasing volume. In addition to the three voices, a programmable Filter is provided for generating complex, dynamic tone colors via subtractive synthesis.

SID allows the microprocessor to read the changing output of the third Oscillator and third Envelope Generator. These outputs can be used as a source of modulation information for creating vibrato, frequency/filter sweeps and similar effects. The third oscillator can also act as a random number generator for games. Two A/D converters are provided for interfacing SID with potentiometers. These can be used for "paddles" in a game environment or as front panel controls in a music synthesizer. SID can process external audio signals, allowing multiple SID chips to be daisy-chained or mixed in complex polyphonic systems.



6581 SID BLOCK DIAGRAM

SID CONTROL REGISTERS

There are 22 eight-bit registers in SID which control the generation of sound. These registers are either WRITE-only or READ-only and are listed below in Table 1.

	ADDRESS A4	A3	A2	A1	A0	REG # (HEX)	DATA D7	D6	D5	D4	D3	D2	D1	D0	REG NAME	REG TYPE
															VOICE 1	
0	0	0	0	0	0	00	F7	F6	F5	F4	F3	F2	F1	F0	FREQ LO	WRITE-only
1	0	0	0	0	1	01	F15	F14	F13	F12	F11	F10	F9	F8	FREQ HI	WRITE-only
2	0	0	0	1	0	02	PW7	PW6	PW5	PW4	PW3	PW2	PW1	PW0	PW LO	WRITE-only
3	0	0	0	1	1	03	—	—	—	—	PW11	PW10	PW9	PW8	PW HI	WRITE-only
4	0	0	1	0	0	04	NOISE	⊓⊓	/|/|	/\/\	TEST	RING MOD	SYNC	GATE	CONTROL REG	WRITE-only
5	0	0	1	0	1	05	ATK3	ATK2	ATK1	ATK0	DCY3	DCY2	DCY1	DCY0	ATTACK/DECAY	WRITE-only
6	0	0	1	1	0	06	STN3	STN2	STN1	STN0	RLS3	RLS2	RLS1	RLS0	SUSTAIN/RELEASE	WRITE-only
															VOICE 2	
7	0	0	1	1	1	07	F7	F6	F5	F4	F3	F2	F1	F0	FREQ LO	WRITE-only
8	0	1	0	0	0	08	F15	F14	F13	F12	F11	F10	F9	F8	FREQ HI	WRITE-only
9	0	1	0	0	1	09	PW7	PW6	PW5	PW4	PW3	PW2	PW1	PW0	PW LO	WRITE-only
10	0	1	0	1	0	0A	—	—	—	—	PW11	PW10	PW9	PW8	PW HI	WRITE-only
11	0	1	0	1	1	0B	NOISE	⊓⊓	/|/|	/\/\	TEST	RING MOD	SYNC	GATE	CONTROL REG	WRITE-only
12	0	1	1	0	0	0C	ATK3	ATK2	ATK1	ATK0	DCY3	DCY2	DCY1	DCY0	ATTACK/DECAY	WRITE-only
13	0	1	1	0	1	0D	STN3	STN2	STN1	STN0	RLS3	RLS2	RLS1	RLS0	SUSTAIN/RELEASE	WRITE-only
															VOICE 3	
14	0	1	1	1	0	0E	F7	F6	F5	F4	F3	F2	F1	F0	FREQ LO	WRITE-only
15	0	1	1	1	1	0F	F15	F14	F13	F12	F11	F10	F9	F8	FREQ HI	WRITE-only
16	1	0	0	0	0	10	PW7	PW6	PW5	PW4	PW3	PW2	PW1	PW0	PW LO	WRITE-only
17	1	0	0	0	1	11	—	—	—	—	PW11	PW10	PW9	PW8	PW HI	WRITE-only
18	1	0	0	1	0	12	NOISE	⊓⊓	/|/|	/\/\	TEST	RING MOD	SYNC	GATE	CONTROL REG	WRITE-only
19	1	0	0	1	1	13	ATK3	ATK2	ATK1	ATK0	DCY3	DCY2	DCY1	DCY0	ATTACK/DECAY	WRITE-only
20	1	0	1	0	0	14	STN3	STN2	STN1	STN0	RLS3	RLS2	RLS1	RLS0	SUSTAIN/RELEASE	WRITE-only
															FILTER	
21	1	0	1	0	1	15	—	—	—	—	—	FC2	FC1	FC0	FC LO	WRITE-only
22	1	0	1	1	0	16	FC10	FC9	FC8	FC7	FC6	FC5	FC4	FC3	FC HI	WRITE-only
23	1	0	1	1	1	17	RES3	RES2	RES1	RES0	FILT EX	FILT 3	FILT 2	FILT 1	RES/FILT	WRITE-only
24	1	1	0	0	0	18	3OFF	HP	BP	LP	VOL3	VOL2	VOL1	VOL0	MODE/VOL	WRITE-only
															MISC.	
25	1	1	0	0	1	19	PX7	PX6	PX5	PX4	PX3	PX2	PX1	PX0	POTX	READ-only
26	1	1	0	1	0	1A	PY7	PY6	PY5	PY4	PY3	PY2	PY1	PY0	POTY	READ-only
27	1	1	0	1	1	1B	O7	O6	O5	O4	O3	O2	O1	O0	OSC3/RANDOM	READ-only
28	1	1	1	0	0	1C	E7	E6	E5	E4	E3	E2	E1	E0	ENV3	READ-only

TABLE 1 - SID REGISTER MAP

SID REGISTER DESCRIPTION

VOICE 1

FREQ LO/FREQ HI (Registers 00,01) — Together these registers form a 16-bit number which linearly controls the Frequency of Oscillator 1. The frequency is determined by the following equation:

$$F_{out} = (F_n * F_{clk}/16777216)\ Hz$$

Where F_n is the 16-bit number in the Frequency registers and F_{clk} is the system clock applied to the Ø2 input (pin 6). For a standard 1.0 MHz clock, the frequency is given by:

$$F_{out} = (F_n * 0.0596)\ Hz$$

A complete table of values for generating 8 octaves of the equally-tempered musical scale with concert A (440 Hz) tuning is provided in Appendix A. It should be noted that the frequency resolution of SID is sufficient for any tuning scale and allows sweeping from note to note (portamento) with no discernable frequency steps.

PW LO/PW HI (Registers 02,03) — Together these registers form a 12-bit number (bits 4-7 of PW HI are not used) which linearly controls the Pulse Width (duty cycle) of the Pulse waveform on Oscillator 1. The pulse width is determined by the following equation:

$$PW_{out} = (PW_n/40.95)\ \%$$

Where PW_n is the 12-bit number in the Pulse Width registers.

The pulse width resolution allows the width to be smoothly swept with no discernable stepping. Note that the Pulse waveform on Oscillator 1 must be selected in order for the Pulse Width registers to have any audible effect. A value of 0 or 4095 ($FFF) in the Pulse Width registers will produce a constant DC output, while a value of 2048 ($800) will produce a square wave.

CONTROL REGISTER (Register 04) — This register contains eight control bits which select various options on Oscillator 1.

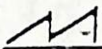
GATE (Bit 0) — The GATE bit controls the Envelope Generator for Voice 1. When this bit is set to a one, the Envelope Generator is Gated (triggered) and the ATTACK/DECAY/SUSTAIN cycle is initiated. When the bit is reset to a zero, the RELEASE cycle begins. The Envelope Generator controls the amplitude of Oscillator 1 appearing at the audio output, therefore, the GATE bit must be set (along with suitable envelope parameters) for the selected output of Oscillator 1 to be audible. A detailed discussion of the Envelope Generator can be found in Appendix B.

SYNC (Bit 1) — The SYNC bit, when set to a one, Synchronizes the fundamental frequency of Oscillator 1 with the fundamental frequency of Oscillator 3, producing "Hard Sync" effects. Varying the frequency of Oscillator 1 with respect to Oscillator 3 produces a wide range of complex harmonic structures from Voice 1 at the frequency of Oscillator 3. In order for sync to occur, Oscillator 3 must be set to some frequency other than zero but preferably lower than the frequency of Oscillator 1. No other parameters of Voice 3 have any effect on sync.

RING MOD (Bit 2) - The RING MOD bit, when set to a one, replaces the Triangle waveform output of Oscillator 1 with a "Ring Modulated" combination of Oscillators 1 and 3. Varying the frequency of Oscillator 1 with respect to Oscillator 3 produces a wide range of non-harmonic overtone structures for creating bell or gong sounds and for special effects. In order for ring modulation to be audible, the Triangle waveform of Oscillator 1 must be selected and Oscillator 3 must be set to some frequency other than zero. No other parameters of Voice 3 have any effect on ring modulation.

TEST (Bit 3) - The TEST bit, when set to a one, resets and locks Oscillator 1 at zero until the TEST bit is cleared. The Noise waveform output of Oscillator 1 is also reset and the Pulse waveform output is held at a DC level. Normally this bit is used for testing purposes; however, it can be used to synchronize Oscillator 1 to external events, allowing the generation of highly complex waveforms under real-time software control.

 (Bit 4) - When set to a one, the Triangle waveform output of Oscillator 1 is selected. The Triangle waveform is low in harmonics and has a mellow, flute-like quality.

 (Bit 5) - When set to a one, the Sawtooth waveform output of Oscillator 1 is selected. The Sawtooth waveform is rich in even and odd harmonics and has a bright, brassy quality.

 (Bit 6) - When set to a one, the Pulse waveform output of Oscillator 1 is selected. The harmonic content of this waveform can be adjusted by the Pulse Width registers, producing tone qualities ranging from a bright, hollow square wave to a nasal, reedy pulse. Sweeping the pulse width in real-time produces a dynamic "phasing" effect which adds a sense of motion to the sound. Rapidly jumping between different pulse widths can produce interesting harmonic sequences.

NOISE (Bit 7) - When set to a one, the Noise output waveform of Oscillator 1 is selected. This output is a random signal which changes at the frequency of Oscillator 1. The sound quality can be varied from a low rumbling to hissing white noise via the Oscillator 1 Frequency registers. Noise is useful in creating explosions, gunshots, jet engines, wind, surf and other unpitched sounds, as well as snare drums and cymbals. Sweeping the oscillator frequency with Noise selected produces a dramatic rushing effect.

One of the output waveforms must be selected for Oscillator 1 to be audible, however, it is NOT necessary to deselect waveforms to silence the output of Voice 1. The amplitude of Voice 1 at the final output is a function of the Envelope Generator only.

NOTE: The oscillator output waveforms are NOT additive. If more than one output waveform is selected simultaneously, the result will be a logical ANDing of the waveforms. Although this technique can be used to generate additional waveforms beyond the four listed above, it must be used with care. If any other waveform is selected while Noise is on, the Noise output can "lock up". If this occurs, the Noise output will remain silent until reset by the TEST bit or by bringing RES (pin 5) low.

ATTACK/DECAY (Register 05) -

Bits 4-7 of this register (ATK0-ATK3) select 1 of 16 ATTACK rates for the Voice 1 Envelope Generator. The ATTACK rate determines how rapidly the output of Voice 1 rises from zero to peak amplitude when the Envelope Generator is Gated. The 16 ATTACK rates are listed below in Table 2.

Bits 0-3 (DCY0-DCY3) select 1 of 16 DECAY rates for the Envelope Generator. The DECAY cycle follows the ATTACK cycle and the DECAY rate determines how rapidly the output falls from the peak amplitude to the selected SUSTAIN level. The 16 DECAY rates are listed in Table 2.

SUSTAIN/RELEASE (Register 06) -

Bits 4-7 of this register (STN0-STN3) select 1 of 16 SUSTAIN levels for the Envelope Generator. The SUSTAIN cycle follows the DECAY cycle and the output of Voice 1 will remain at the selected SUSTAIN amplitude as long as the Gate bit remains set. The SUSTAIN levels range from zero to peak amplitude in 16 linear steps, with a SUSTAIN value of 0 selecting zero amplitude and a SUSTAIN value of 15 ($F) selecting the peak amplitude. A SUSTAIN value of 8 would cause Voice 1 to SUSTAIN at an amplitude one-half the peak amplitude reached by the ATTACK cycle.

Bits 0-3 (RLS0-RLS3) select 1 of 16 RELEASE rates for the Envelope Generator. The RELEASE cycle follows the SUSTAIN cycle when the Gate bit is reset to zero. At this time, the output of Voice 1 will fall from the SUSTAIN amplitude to zero amplitude at the selected RELEASE rate. The 16 RELEASE rates are identical to the DECAY rates.

> NOTE: The cycling of the Envelope Generator can be altered at any point via the Gate bit. The Envelope Generator can be Gated and Released without restriction. For example, if the Gate bit is reset before the envelope has finished the ATTACK cycle, the RELEASE cycle will immediately begin, starting from whatever amplitude had been reached. If the envelope is then Gated again (before the RELEASE cycle has reached zero amplitude), another ATTACK cycle will begin, starting from whatever amplitude had been reached. This technique can be used to generate complex amplitude envelopes via real-time software control.

TABLE 2 - ENVELOPE RATES

VALUE DEC	(HEX)	ATTACK RATE (Time/Cycle)	DECAY/RELEASE RATE (Time/Cycle)
0	(0)	2 mS	6 mS
1	(1)	8 mS	24 mS
2	(2)	16 mS	48 mS
3	(3)	24 mS	72 mS
4	(4)	38 mS	114 mS
5	(5)	56 mS	168 mS
6	(6)	68 mS	204 mS
7	(7)	80 mS	240 mS
8	(8)	100 mS	300 mS
9	(9)	250 mS	750 mS
10	(A)	500 mS	1.5 S
11	(B)	800 mS	2.4 S
12	(C)	1 S	3 S
13	(D)	3 S	9 S
14	(E)	5 S	15 S
15	(F)	8 S	24 S

NOTE: Envelope rates are based on a 1.0 MHz 02 clock. For other 02 frequencies, multiply the given rate by 1 MHz/02. The rates refer to the amount of time per cycle. For example, given an ATTACK value of 2, the ATTACK cycle would take 16 mS to rise from zero to peak amplitude. The DECAY/RELEASE rates refer to the amount of time these cycles would take to fall from peak amplitude to zero.

Registers 07-$0D control Voice 2 and are functionally identical to registers 00-06 with these exceptions:

When selected, SYNC synchronizes Oscillator 2 with Oscillator 1.
When selected, RING MOD replaces the Triangle output of Oscillator 2 with the ring modulated combination of Oscillators 2 and 1.

VOICE 3

Registers $0E-$14 control Voice 3 and are functionally identical to registers 00-06 with these exceptions:

When selected, SYNC synchronizes Oscillator 3 with Oscillator 2.
When selected, RING MOD replaces the Triangle output of Oscillator 3 with the ring modulated combination of Oscillators 3 and 2.

Typical operation of a voice consists of selecting the desired parameters: frequency, waveform, effects (SYNC, RING MOD) and envelope rates, then gating the voice whenever the sound is desired. The sound can be sustained for any length of time and terminated by clearing the Gate bit. Each voice can be used separately, with independent parameters and gating, or in unison to create a single, powerful voice. When used in unison, a slight detuning of each oscillator or tuning to musical intervals creates a rich, animated sound.

FILTER

FC LO/FC HI (Registers $15,$16) -

Together these registers form an 11-bit number (bits 3-7 of FC LO are not used) which linearly controls the Cutoff (or Center) Frequency of the programmable Filter. The approximate Cutoff Frequency is determined by the following equation:

$$FCout = ((6.6E-8 + FCn * 1.28E-8)/C) \text{ Hz}$$

Where FCn is the 11-bit number in the Cutoff registers and C is the value of the two Filter capacitors connected to pins 1-4. For the recommended capacitor value of 2200 pF, the approximate range of the Filter is 30 Hz-12 KHz according to the following equation:

$$FCout = (30 + FCn * 5.8) \text{ Hz}$$

The frequency range of the Filter can be altered to suit specific applications. Refer to the Pin Description section for more information.

RES/FILT (Register $17) -

Bits 4-7 of this register (RES0-RES3) control the Resonance of the Filter. Resonance is a peaking effect which emphasizes frequency components at the Cutoff Frequency of the Filter, causing a sharper sound. There are 16 Resonance settings ranging linearly from no resonance (0) to maximum resonance (15 or $F).

Bits 0-3 determine which signals will be routed through the Filter:

FILT 1 (Bit 0) - When set to a zero, Voice 1 appears directly at the audio output and the Filter has no effect on it. When set to a one, Voice 1 will be processed through the Filter and the harmonic content of Voice 1 will be altered according to the selected Filter parameters.

FILT 2 (Bit 1) - Same as bit 0 for Voice 2.

FILT 3 (Bit 2) - Same as bit 0 for Voice 3.

FILTEX (Bit 3) - Same as bit 0 for External audio input (pin 26).

MODE/VOL (Register $18) -

Bits 4-7 of this register select various Filter mode and output options:

LP (Bit 4) - When set to a one, the Low Pass output of the Filter is selected and sent to the audio output. For a given Filter input signal, all frequency components below the Filter Cutoff Frequency are passed unaltered, while all frequency components above the Cutoff are attenuated at a rate of 12 dB/Octave. The Low Pass mode produces full-bodied sounds.

BP (Bit 5) - Same as bit 4 for the Band Pass output. All frequency components above and below the Cutoff are attenuated at a rate of 6 dB/Octave. The Band Pass mode produces thin, open sounds.

HP (Bit 6) - Same as bit 4 for the High Pass output. All frequency components above the Cutoff are passed unaltered, while all frequency components below the Cutoff are attenuated at a rate of 12 dB/Octave. The High Pass mode produces tinny, buzzy sounds.

3 OFF (Bit 7) - When set to a one, the output of Voice 3 is disconnected from the direct audio path. Setting Voice 3 to bypass the Filter (FILT 3=0) and setting 3 OFF to a one prevents Voice 3 from reaching the audio output. This allows Voice 3 to be used for modulation purposes without any undesirable output. More information on modulation effects can be found in Appendix C.

NOTE: The Filter output modes ARE additive and multiple Filter modes may be selected simultaneously. For example, both LP and HP modes can be selected to produce a Notch (or Band Reject) Filter response. In order for the Filter to have any audible effect, at least one Filter output must be selected and and at least one Voice must be routed through the Filter. The Filter is, perhaps, the most important element in SID as it allows the generation of complex tone colors via subtractive synthesis (the Filter is used to eliminate specific frequency components from a harmonically-rich input signal). The best results are achieved by varying the Cutoff Frequency in real-time. Further discussion of the Filter appears in Appendix C.

Bits 0-3 (VOL0-VOL3) select 1 of 16 overall Volume levels for the final composite audio output. The output volume levels range from no output (0) to maximum volume (15 or $F) in 16 linear steps. This control can be used as a static volume control for balancing levels in multi-chip systems or for creating dynamic volume effects, such as Tremolo. Some Volume level other than zero must be selected in order for SID to produce any sound.

MISC

POTX (Register $19) -

This register allows the microprocessor to read the position of the potentiometer tied to POTX (pin 24), with values ranging from 0 at minimum resistance, to 255 ($FF) at maximum resistance. The value is always valid and is updated every 512 Ø2 clock cycles. See the Pin Description section for information on pot and capacitor values.

POTY (Register $1A) -

Same as POTX for the pot tied to POTY (pin 23).

OSC 3/RANDOM (Register $1B) -

This register allows the microprocessor to read the upper 8 output bits of Oscillator 3. The character of the numbers generated is directly related to the waveform selected. If the Sawtooth waveform of Oscillator 3 is selected, this register will present a series of numbers incrementing from 0 to 255 ($FF) at a rate determined by the frequency of Oscillator 3. If the Triangle waveform is selected, the output will increment from 0 up to 255, then decrement down to 0. If the Pulse waveform is selected, the output will jump between 0 and 255. Selecting the Noise waveform will produce a series of random numbers, therefore, this register can be used as a random number generator for games. There are numerous timing and sequencing applications for the OSC 3 register, however, the chief function is probably that of a modulation generator. The numbers generated by this register can be added, via software, to the Oscillator or Filter Frequency registers or the Pulse Width registers in real-time. Many dynamic effects can be generated in this manner. Siren-like sounds can be created by adding the OSC 3 Sawtooth output to the frequency control of another oscillator. Synthesizer "Sample and Hold" effects can be produced by adding the OSC 3 Noise output to the Filter Frequency control registers. Vibrato can be produced by setting Oscillator 3 to a frequency around 7 Hz and adding the OSC 3 Triangle output (with proper scaling) to the Frequency control of another oscillator. An unlimited range of effects are available by altering the frequency of Oscillator 3 and scaling the OSC 3 output. Normally, when Oscillator 3 is used for modulation, the audio output of Voice 3 should be eliminated (3 OFF=1).

ENV 3 (Register $1C) -

Same as OSC 3, but this register allows the microprocessor to read the output of the Voice 3 Envelope Generator. This output can be added to the Filter Frequency to produce harmonic envelopes, WAH WAH, and similar effects. "Phaser" sounds can be created by adding this output to the frequency control registers of an oscillator. The Voice 3 Envelope Generator must be Gated in order to produce any output from this register. The OSC 3 register, however, always reflects the changing output of the oscillator and is not affected in any way by the Envelope Generator. Further information on modulation can be found in Appendix C.

SID PIN DESCRIPTION

CAP1A,CAP1B (Pins 1,2),CAP2A,CAP2B (Pins 3,4) - These pins are used to connect the two integrating capacitors required by the programmable Filter. C1 connects between pins 1 and 2, C2 between pins 3 and 4. Both capacitors should be the same value. Normal operation of the Filter over the audio range (approximately 30 Hz-12 kHz) is accomplished with a value of 2200 pF for C1 and C2. Polystyrene capacitors are preferred and in complex polyphonic systems, where many SID chips must track each other, matched capacitors are recommended. The frequency range of the Filter can be tailored to specific applications by the choice of capacitor values. For example, a low-cost game may not require full high-frequency response. In this case, larger values for C1 and C2 could be chosen to provide more control over the bass frequencies of the Filter. The maximum Cutoff Frequency of the Filter is given by:

FCmax = 2.6E-5/C

Where C is the capacitor value. The range of the Filter extends 9 octaves below the maximum Cutoff Frequency.

$\overline{\text{RES}}$ (Pin 5) - This TTL-level input is the reset control for SID. When brought low for at least ten Ø2 cycles, all internal registers are reset to zero and the audio output is silenced. This pin is normally connected to the reset line of the microprocessor or a power-on-clear circuit.

Ø2 (Pin 6) - This TTL-level input is the master clock for SID. All oscillator frequencies and envelope rates are referenced to this clock. Ø2 also controls data transfers between SID and the microprocessor. Data can only be transferred when Ø2 is high. Essentially, Ø2 acts as a high-active chip select as far as data tranfers are concerned. This pin is normally connected to the system clock, with a nominal operating frequency of 1.0 MHz.

R/$\overline{\text{W}}$ (Pin 7) - This TTL-level input controls the direction of data transfers between SID and the microprocessor. If the chip select conditions have been met, a high on this line allows the microprocessor to Read data from the selected SID register and a low allows the microprocessor to Write data into the selected SID register. This pin is normally connected to the system Read/Write line.

$\overline{\text{CS}}$ (Pin 8) - This TTL-level input is a low active chip select which controls data tranfers between SID and the microprocessor. $\overline{\text{CS}}$ must be low for any transfer. A Read from the selected SID register can only occur if $\overline{\text{CS}}$ is low, Ø2 is high and R/$\overline{\text{W}}$ is high. A Write to the selected SID register can only occur if $\overline{\text{CS}}$ is low, Ø2 is high and R/$\overline{\text{W}}$ is low. This pin is normally connected to address decoding circuitry, allowing SID to reside in the memory map of a system.

A0-A4 (Pins 9-13) - These TTL-level inputs are used to select one of the 29 SID registers. Although enough addresses are provided to select 1 of 32 registers, the remaining three register locations are not used. A Write to any of these three locations is ignored and a Read returns invalid data. These pins are normally connected to the corresponding address lines of the microprocessor so that SID may be addressed in the same manner as memory.

GND (Pin 14) - For best results, the ground line between SID and the power supply should be separate from ground lines to other digital circuitry. This will minimize digital noise at the audio output.

-D7 (Pins 15-22) - These bidirectional lines are used to transfer data
etween SID and the microprocessor. They are TTL compatible in the
nput mode and capable of driving 2 TTL loads in the output mode. The
ata buffers are usually in the high-impedance off state. During
write operation, the data buffers remain in the off (input) state and the
icroprocessor supplies data to SID over these lines. During a Read
eration, the data buffers turn on and SID supplies data to the micro-
ocessor over these lines. The pins are normally connected to the
orresponding data lines of the microprocessor.

OTX,POTY (Pins 24,23) - These pins are inputs to the A/D converters used
o digitize the position of potentiometers. The conversion process is
ased on the time constant of a capacitor tied from the POT pin to
round, charged by a potentiometer tied from the POT pin to +5 volts.
he component values are determined by:

RC = 4.7E-4

Where R is the maximum resistance of the pot and C is the capacitor.
The larger the capacitor, the smaller the POT value jitter.
The recommended values for R and C are 470 KOhms and 1000 pF.
Note that a separate pot and cap are required for each POT pin.

cc (Pin 25) - As with the GND line, a separate +5 VDC line should be run
etween SID Vcc and the power supply in order to minimize noise. A bypass
apacitor should be located close to the pin.

XT IN (Pin 26) - This analog input allows external audio signals to be mixed
ith the audio output of SID or processed through the Filter. Typical
ources include voice, guitar and organ. The input impedance of this pin is
n the order of 100 KOhms. Any signal applied directly to the pin should ride
a DC level of 6 volts and should not exceed 3 volts p-p. In order to
vent any inteference caused by DC level differences, external signals
uld be AC-coupled to EXT IN by an electrolytic capacitor in the 1-10 uF
ange. As the direct audio path (FILTEX=0) has unity gain, EXT IN can be
sed to mix outputs of many SID chips by daisy-chaining. The number of chips
hat can be chained in this manner is determined by the amount of noise and
istortion allowable at the final output. Note that the output Volume
ontrol will affect not only the three SID voices, but also any external
nputs.

UDIO OUT (Pin 27) - This open-source buffer is the final audio output
f SID, comprised of the three SID voices, the Filter and any
xternal input. The output level is set by the output Volume control and
eaches a maximum of 2 volts p-p at a DC level of 6 volts. A source
esistor from AUDIO OUT to ground is required for proper operation.
he recommended resistance is 1 kOhm for a standard output impedance. As
he output of SID rides at a 6 volt DC level, it should be AC-coupled to
ny audio amplifier with an electrolytic capacitor in the 1-10 uF range.

dd (Pin 28) - As with Vcc, a separate +12 VDC line should be
un to SID Vdd and a bypass capacitor should be used.

6581 SID CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS

RATING	SYMBOL	VALUE	UNITS
Supply Voltage	Vdd	-0.3 to +17	VDC
Supply Voltage	Vcc	-0.3 to +7	VDC
Input Voltage (analog)	Vina	-0.3 to +17	VDC
Input Voltage (digital)	Vind	-0.3 to +7	VDC
Operating Temperature	Ta	0 to +70	°C
Storage Temperature	Tstg	-55 to +150	°C

ELECTRICAL CHARACTERISTICS (Vdd=12 ± 5% VDC, Vcc=5 ± 5% VDC, Ta=0 to 70°C)

CHARACTERISTIC		SYMBOL	MIN	TYP	MAX	UNITS
Input High Voltage	($\overline{RES}$, 02, R/$\overline{W}$, $\overline{CS}$, A0-A4, D0-D7)	Vih	2	-	Vcc	VDC
Input Low Voltage		Vil	-0.3	-	0.8	VDC
Input Leakage Current	($\overline{RES}$, 02, R/$\overline{W}$, $\overline{CS}$, A0-A4; Vin=0-5 VDC)	Iin	-	-	2.5	uA
Three-State (Off) Input Leakage Current	(D0-D7; Vcc=max, Vin=0.4-2.4 VDC)	Itsi	-	-	10	uA
Output High Voltage	(D0-D7; Vcc=min, I load=200 uA)	Voh	2.4	-	Vcc-0.7	VDC
Output Low Voltage	(D0-D7; Vcc=max, I load=3.2 mA)	Vol	GND	-	0.4	VDC
Output High Current	(D0-D7; Sourcing, Voh=2.4 VDC)	Ioh	200	-	-	uA
Output Low Current	(D0-D7; Sinking, Vol=0.4 VDC)	Iol	3.2	-	-	mA
Input Capacitance	($\overline{RES}$, 02, R/$\overline{W}$, $\overline{CS}$, A0-A4, D0-D7)	Cin	-	-	10	pF
Pot Trigger Voltage	(POTX, POTY)	Vpot	-	Vcc/2	-	VDC
Pot Sink Current	(POTX, POTY)	Ipot	500	-	-	uA
Input Impedance	(EXT IN)	Rin	100	150	-	KOhms
Audio Input Voltage	(EXT IN)	Vin	5.7 -	6 0.5	6.3 3	VDC VAC
Audio Output Voltage	(AUDIO OUT; 1 KOhm load, volume=max) One Voice on: All Voices on:	Vout	5.7 0.4 1.0	6 0.5 1.5	6.3 0.6 2.0	VDC VAC VAC
Power Supply Current	(Vdd)	Idd	-	20	25	mA
Power Supply Current	(Vcc)	Icc	-	70	100	mA
Power Dissipation	(Total)	Pd	-	600	1000	mW

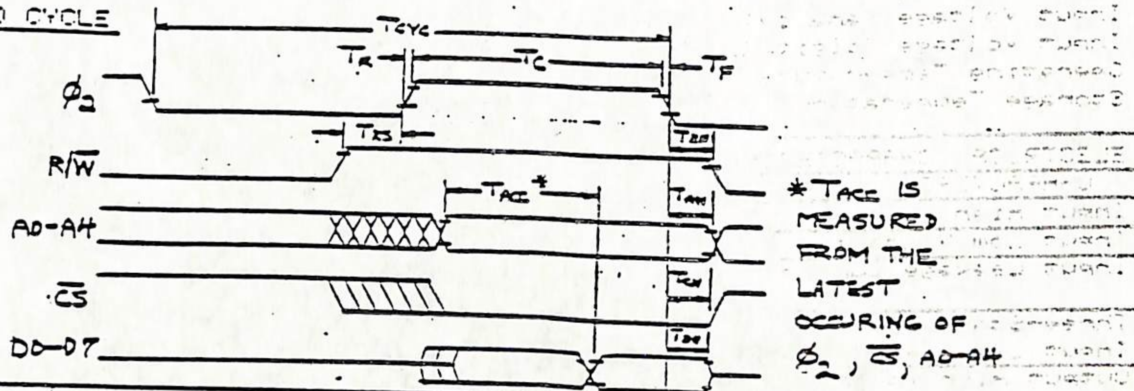
COMMENT

Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only. Functional operation of this device at these or any other conditions above those indicated in the operational sections of this specification is not implied and exposure to absolute maximum rating conditions for extended periods may affect device reliability.

All inputs contain protection circuitry to prevent damage due to high static discharges. Care should be exercised to prevent unnecessary application of voltages in excess of the allowable limits.

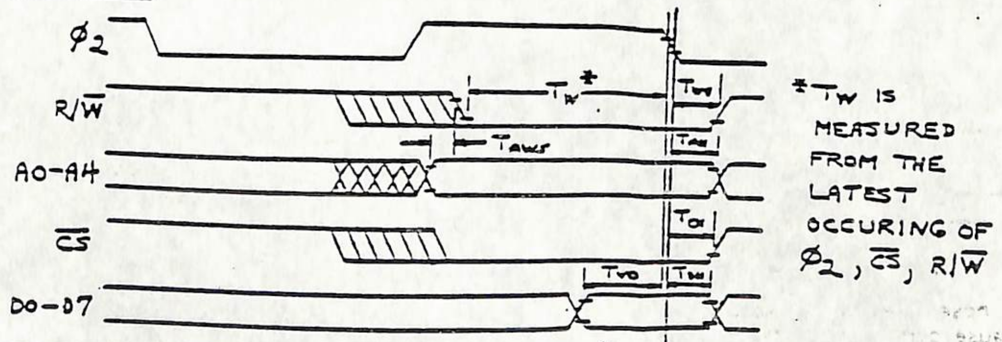
6581 SID TIMING

READ CYCLE



SYMBOL	NAME	MIN	TYP	MAX	UNITS
Tcyc	Clock Cycle Time	1	-	20	uS
Tc	Clock High Pulse Width	450	500	10,000	nS
Tr,Tf	Clock Rise/Fall Time	-	-	25	nS
Trs	Read Set-up Time	0	-	-	nS
Trh	Read Hold Time	0	-	-	nS
Tacc	Access Time	-	-	300	nS
Tah	Address Hold Time	10	-	-	nS
Tch	Chip Select Hold Time	0	-	-	nS
Tdh	Data Hold Time	20	-	-	nS

WRITE CYCLE



SYMBOL	NAME	MIN	TYP	MAX	UNITS
Tw	Write Pulse Width	350	-	-	nS
Twh	Write Hold Time	0	-	-	nS
Taws	Address Set-up Time	0	-	-	nS
Tah	Address Hold Time	10	-	-	nS
Tch	Chip Select Hold Time	0	-	-	nS
Tvd	Valid Data	80	-	-	nS
Tdh	Data Hold Time	10	-	-	nS

APPENDIX A - EQUAL-TEMPERED MUSICAL SCALE VALUES

The following table lists the numerical values which must be stored in the SID Oscillator frequency control registers to produce the notes of the equal-tempered musical scale. The equal-tempered scale consists of an octave containing 12 semitones (notes): C,D,E,F,G,A,B and C#,D#,F#,G#,A#. Th frequency of each semitone is exactly the 12th root of 2 ($\sqrt[12]{2}$) times the frequency of the previous semitone. The table is based on a Ø2 clock of 1.0 Mhz. Refer to the equation given in the Register Description for use of othe master clock frequencies. The scale selected is concert pitch, in which A4 = 440 Hz. Transpositions of this scale and scales other than the equal-tempered scale are also possible.

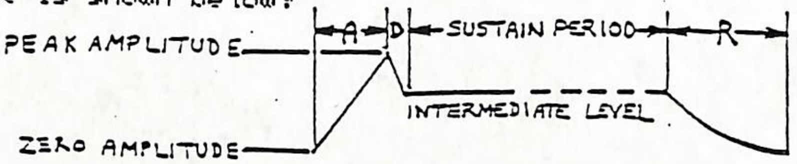
	MUSICAL NOTE	FREQ (HZ)	OSC F_n (Decimal)	OSC F_n (HEX)
0	C0	16.35	269	010D
1	C0#	17.32	285	011D
2	D0	18.35	302	012E
3	D0#	19.45	320	0140
4	E0	20.60	339	0153
5	F0	21.83	359	0167
6	F0#	23.12	380	017C
7	G0	24.50	403	0193
8	G0#	25.96	427	01AB
9	A0	27.50	452	01C4
10	A0#	29.14	479	01DF
11	B0	30.87	508	01FC
12	C1	32.70	538	021A
13	C1#	34.65	570	023A
14	D1	36.71	604	025C
15	D1#	38.89	640	0280
16	E1	41.20	678	02A6
17	F1	43.65	718	02CE
18	F1#	46.25	761	02F9
19	G1	49.00	806	0326
20	G1#	51.91	854	0356
21	A1	55.00	905	0389
22	A1#	58.27	958	03BE
23	B1	61.74	1015	03F7
24	C2	65.41	1076	0434
25	C2#	69.30	1140	0474
26	D2	73.42	1208	04B8
27	D2#	77.78	1279	04FF
28	E2	82.41	1355	054B
29	F2	87.31	1436	059C
30	F2#	92.50	1521	05F1
31	G2	98.00	1612	064C
32	G2#	103.83	1708	06AC
33	A2	110.00	1809	0711
34	A2#	116.54	1917	077D
35	B2	123.47	2031	07EF
36	C3	130.81	2152	0868
37	C3#	138.59	2280	08E8
38	D3	146.83	2415	096F
39	D3#	155.56	2559	09FF
40	E3	164.81	2711	0A97

41	F3	174.61	2872	0B38
42	F3#	185.00	3043	0BE3
43	G3	196.00	3224	0C98
44	G3#	207.65	3416	0D58
45	A3	220.00	3619	0E23
46	A3#	233.08	3834	0EFA
47	B3	246.94	4062	0FDE
48	C4	261.63	4303	10CF
49	C4#	277.18	4559	11CF
50	D4	293.66	4830	12DE
51	D4#	311.13	5117	13FD
52	E4	329.63	5422	152E
53	F4	349.23	5744	1670
54	F4#	369.99	6086	17C6
55	G4	392.00	6448	1930
56	G4#	415.30	6831	1AAF
57	A4	440.00	7237	1C45
58	A4#	466.16	7668	1DF4
59	B4	493.88	8124	1FBC
60	C5	523.25	8607	219F
61	C5#	554.37	9118	239E
62	D5	587.33	9661	25BD
63	D5#	622.25	10235	27FB
64	E5	659.26	10844	2A5C
65	F5	698.46	11488	2CE0
66	F5#	739.99	12172	2F8C
67	G5	783.99	12895	325F
68	G5#	830.61	13662	355E
69	A5	880.00	14474	388A
70	A5#	932.33	15335	3BE7
71	B5	987.77	16247	3F77
72	C6	1046.50	17213	433D
73	C6#	1108.73	18237	473D
74	D6	1174.66	19321	4B79
75	D6#	1244.51	20470	4FF6
76	E6	318.51	21687	54B7
77	F6	1396.91	22977	59C1
78	F6#	1479.98	24343	5F17
79	G6	1567.98	25791	64BF
80	G6#	1661.22	27324	6ABC
81	A6	1760.00	28949	7115
82	A6#	1864.66	30670	77CE
83	B6	1975.53	32494	7EEE
84	C7	2093.00	34426	867A
85	C7#	2217.46	36473	8E79
86	D7	2349.32	38642	96F2
87	D7#	2489.02	40940	9FEC
88	E7	2637.02	43374	A96E
89	F7	2793.83	45954	B382
90	F7#	2959.96	48686	BE2E
91	G7	3135.96	51581	C97D
92	G7#	3322.44	54648	D578
93	A7	3520.00	57898	E22A
94	A7#	3729.31	61341	EF9D
95	B7	3951.07	64988	FDDC

Although the table above provides a simple and quick method for generating the equal-tempered scale, it is very memory inefficient as it requires 192 bytes for the table alone. Memory efficiency can be improved by determining the note value algorithmically. Using the fact that each note in an octave is exactly half the frequency of that note in the next octave, the note look-up table can be reduced from 96 entries to 12 entries, as there are 12 notes per octave. If the 12 entries (24 bytes) consist of the 16-bit value for the eighth octave (C7 through B7), then notes in lower octaves can be derived by choosing the appropriate note in the eighth octave and dividing the 16-bit value by two for each octave of difference. As division by two is nothing more than a right-shift of the value, the calculation can easily be accomplished by a simple software routine. Although note B7 is beyond the range of the Oscillators, this value should still be included in the table for calculation purposes (the MSB of B7 would require a special software case, such as generating this bit in the CARRY before shifting). Each note must be specified in a form which indicates which of the 12 semitones is desired, and which of the eight octaves the semitone is in. Since four bits are necessary to select 1 of 12 semitones and three bits are necessary to select 1 of 8 octaves, the information can fit in one byte, with the lower nybble selecting the semitone (by addressing the look-up table) and the upper nybble being used by the division routine to determine how many times the table value must be right-shifted.

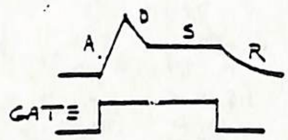
APPENDIX B - SID ENVELOPE GENERATORS

The four-part ADSR (ATTACK, DECAY, SUSTAIN, RELEASE) envelope generator has been proven in electronic music to provide the optimum trade-off bewteen flexibility and ease of amplitude control. Appropriate selection of envelope parameters allows the simulation of a wide range of percussion and sustained instruments. The violin is a good example of a sustained instrument. The violinist controls the volume by bowing the instrument. Typically, the volume builds slowly, reaches a peak, then drops to an intermediate level. The violinist can maintain this level for as long as desired, then the volume is allowed to slowly die away. A "snapshot" of this envelope is shown below:



This volume envelope can be easily reproduced by the ADSR as shown below, with typical envelope rates:

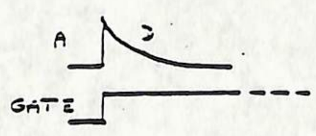
ATTACK:	10 (\$A)	500 mS
DECAY:	8	300 mS
SUSTAIN:	10 (\$A)	
RELEASE:	9	750 mS



Note that the tone can be held at the intermediate SUSTAIN level for as long as desired. The tone will not begin to die away until GATE is cleared. With minor alterations, this basic envelope can be used for brass and woodwinds as well as strings.

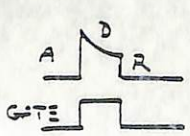
An entirely different form of envelope is produced by percussion instruments such as drums, cymbals and gongs, as well as certain keyboards such as pianos and harpsichords. The percussion envelope is characterized by a nearly instantaneous attack, immediately followed by a decay to zero volume. Percussion instruments cannot be sustained at a constant amplitude. For example, the instant a drum is struck, the sound reaches full volume and decays rapidly regardless of how it was struck. A typical cymbal envelope is shown below:

ATTACK:	0	2 mS
DECAY:	9	750 mS
SUSTAIN:	0	
RELEASE:	9	750 mS



Note that the tone immediately begins to decay to zero amplitude after the peak is reached, regardless of when GATE is cleared. The amplitude envelope of pianos and harpsichords is somewhat more complicated, but can be generated quite easily with the ADSR. These instruments reach full volume when a key is first struck. The amplitude immediately begins to die away slowly as long as the key remains depressed. If the key is released before the sound has fully died away, the amplitude will immediately drop to zero. This envelope is shown below:

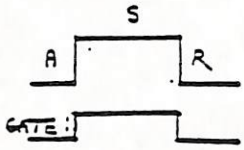
ATTACK:	0	2 mS
DECAY:	9	750 mS
SUSTAIN:	0	
RELEASE:	0	6 mS



Note that the tone decays slowly until GATE is cleared, at which point the amplitude drops rapidly to zero.

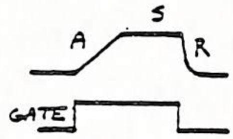
The most simple envelope is that of the organ. When a key is pressed, the tone immediately reaches full volume and remains there. When the key is released, the tone drops immediately to zero volume. This envelope is shown below:

ATTACK: 0 2 mS
DECAY: 0 6 mS
SUSTAIN: 15 ($F)
RELEASE: 0 6 mS

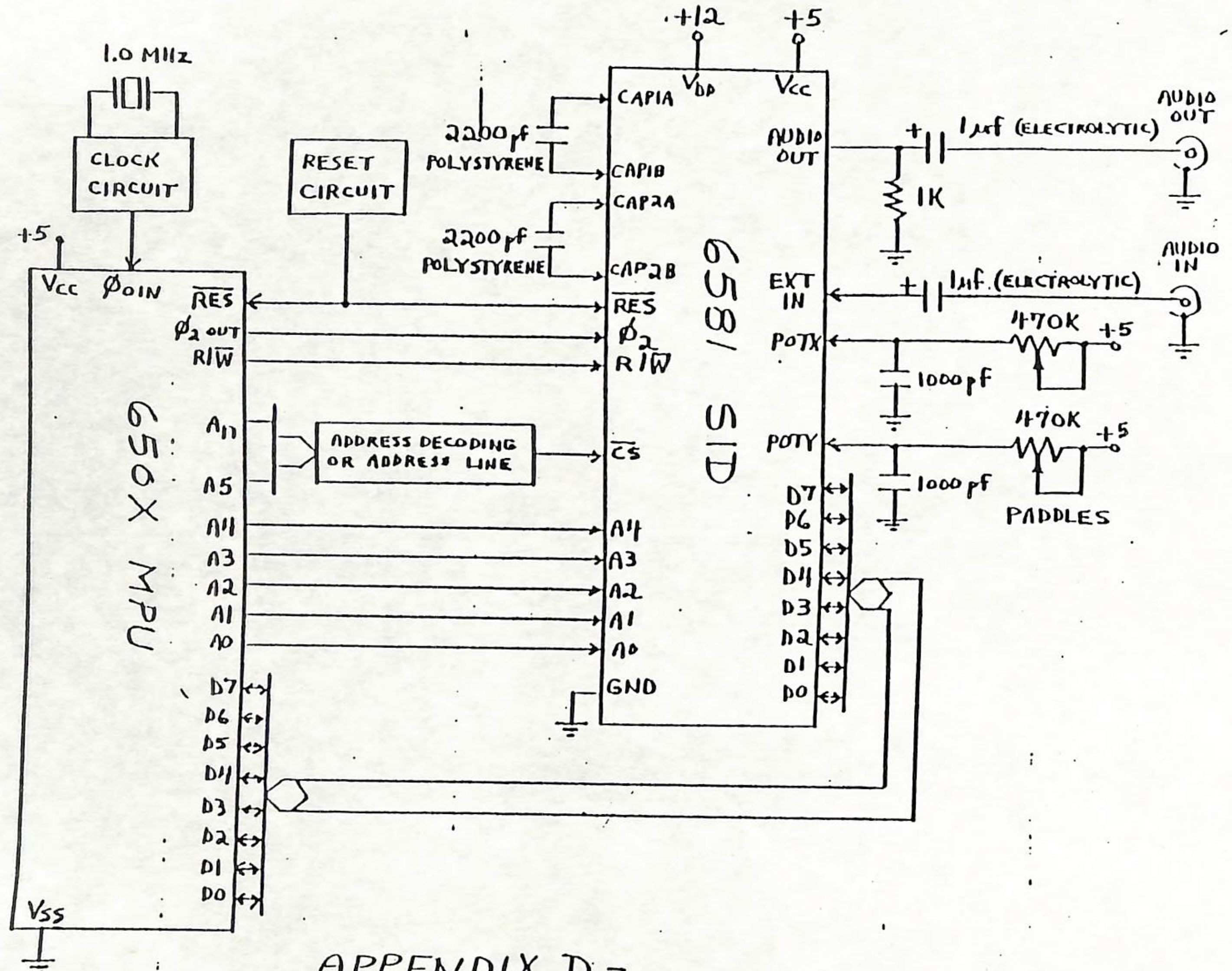


The real power of SID lies in the ability to create original sounds rather than simulations of acoustic instruments. The ADSR is capable of creating envelopes which do not correspond to any "real" instruments. A good example would be the "backwards" envelope. This envelope is characterized by a slow attack and rapid decay which sounds very much like an instrument that has been recorded on tape then played backwards. This envelope is shown below

ATTACK: 10 ($A) 500 mS
DECAY: 0 6 mS
SUSTAIN: 15 ($F)
RELEASE: 3 72 mS



Many unique sounds can be created by applying the amplitude envelope of one instrument to the harmonic structure of another. This produces sounds similar to familiar acoustic instruments, yet notably different. In general, sound is quite subjective and experimentation with various envelope rates and harmonic contents will be necessary in order to achieve the desired sound.



APPENDIX D -
TYPICAL 6581 SID APPLICATION

SECTION III

Commodore 64 Memory Maps

COMMODORE 64 MEMORY MAP (*=USEFUL MEMORY LOCATION)

HEX	DECIMAL	DESCRIPTION
0000	0	6510 Data-direction register
0001	1	6510 Output register
0002	2	Unused
0003-0004	3-4	Float-Fixed vector
0005-0006	5-6	Fixed-Float vector
0007	7	Search character
0008	8	Scan-quotes flag
0009	9	TAB column save
000A	10	0=LOAD, 1=VERIFY
000B	11	Input buffer pointer/# subscript
000C	12	Default DIM flag
000D	13	Type: FF=string, 00=numeric
000E	14	Type: 80=integer, 00=floating point
000F	15	DATA scan/LIST quote/memory flag
0010	16	Subscript/FNx flag
0011	17	0=INPUT;$40=GET;$98=READ
0012	18	ATN sign/Comparison eval flag
0013	19	Current I/O prompt flag
*0014-0015	20-21	Integer value
0016	22	Pointer: temporary string stack
0017-0018	23-24	Last temp string vector
0019-0021	25-33	Stack for temporary strings
0022-0025	34-37	Utility pointer area
0026-002A	38-42	Product area for multiplication
*002B-002C	43-44	Pointer: Start of Basic
*002D-002E	45-46	Pointer: Start of Variables
*002F-0030	47-48	Pointer: Start of Arrays
*0031-0032	49-50	Pointer: End of Arrays
*0033-0034	51-52	Pointer: String storage (moving down)
0035-0036	53-54	Utility string pointer
*0037-0038	55-56	Pointer: Limit of memory
0039-003A	57-58	Current Basic line number
003B-003C	59-60	Previous Basic line number
003D-003E	61-62	Pointer: Basic statement for CONT
003F-0040	63-64	Current DATA line number
0041-0042	65-66	Current DATA address
*0043-0044	67-68	Input vector
0045-0046	69-70	Current variable name
0047-0048	71-72	Current variable address
0049-004A	73-74	Variable pointer for FOR/NEXT
004B-004C	75-76	Y-save; op-save; Basic pointer save
004D	77	Comparison symbol accumulator
004E-0053	78-83	Misc work area, pointers, etc
0054-0056	84-86	Jump vector for functions
0057-0060	87-96	Misc numeric work area
*0061	97	Accum#1: Exponent
*0062-0065	98-101	Accum#1: Mantissa
*0066	102	Accum#1: Sign
0067	103	Series evaluation constant pointer
0068	104	Accum#1 hi-order (overflow)

HEX	DECIMAL	DESCRIPTION
0069-006E	105-110	Accum#2: Exponent, etc.
006F	111	Sign comparison, Acc#1 vs #2
0070	112	Accum#1 lo-order (rounding)
0071-0072	113-114	Cassette buffer length/Series pointer
*0073-008A	115-138	CHRGET subroutine (get BASIC char)
007A-007B	122-123	Basic pointer (within subroutine)
008B-008F	139-143	RND seed value
*0090	144	Status word ST
0091	145	Keyswitch PIA: STOP and RVS flags
0092	146	Timing constant for tape
0093	147	Load=0, Verify=1
0094	148	Serial output: deferred char flag
0095	149	Serial deferred character
0096	150	Tape EOT received
0097	151	Register save
*0098	152	How many open files
*0099	153	Input device (normally 0)
*009A	154	Output (CMD) device, normally 3
009B	155	Tape character parity
009C	156	Byte-received flag
009D	157	Direct=$80/RUN=0 output control
009E	158	Tape Pass 1 error log/char buffer
009F	159	Tape Pass 2 error log corrected
*00A0-00A2	160-162	Jiffy Clock (HML)
00A3	163	Serial bit count/EOI flag
00A4	164	Cycle count
00A5	165	Countdown,tape write/bit count
00A6	166	Pointer: tape buffer
00A7	167	Tape Write ldr count/Read pass/inbit
00A8	168	Tape Write new byte/Read error/inbit cnt
00A9	169	Write start bit/Read bit err/stbit
00AA	170	Tape Scan;Cnt;Ld;End/byte assy
00AB	171	Write lead length/Rd checksum/parity
00AC-00AD	172-173	Pointer: tape buffer, scrolling
00AE-00AF	174-175	Tape end addresses/End of program
00B0-00B1	176-177	Tape timing constants
*00B2-00B3	178-179	Pointer: start of tape buffer
00B4	180	Tape timer (1=enable); bit cnt
00B5	181	Tape EOT/RS232 next bit to send
00B6	182	Read character error/outbyte buffer
*00B7	183	# characters in file name
*00B8	184	Current logical file
*00B9	185	Current secondary address
*00BA	186	Current device
*00BB-00BC	187-188	Pointer: to file name
00BD	189	Write shift word/Read input char
00BE	190	# blocks remaining to Write/Read
00BF	191	Serial word buffer
00C0	192	Tape motor interlock
00C1-00C2	193-194	I/O start addresses
00C3-00C4	195-196	Kernel setup pointer
*00C5	197	Current key pressed

HEX	DECIMAL	DESCRIPTION
*00C6	198	# chars in keyboard buffer
*00C7	199	Screen reverse flag
00C8	200	Pointer: End-of-line for input
00C9--00CA	201--202	Input cursor log (row, column)
*00CB	203	Which key: 64 if no key
00CC	204	cursor enable (0=flash cursor)
00CD	205	Cursor timing countdown
00CE	206	Character under cursor
00CF	207	Cursor in blink phase
00D0	208	Input from screen/from keyboard
*00D1-00D2	209-210	Pointer to screen line
*00D3	211	Position of cursor on above line
00D4	212	0=direct cursor, else programmed
*00D5	213	Current screen line length
*00D6	214	Row where cursor lives
00D7	215	Last inkey/checksum/buffer
*00D8	216	# of INSERTs outstanding
*00D9-00F0	217-240	Screen line link table
00F1	241	Dummy screen link
00F2	242	Screen row marker
*00F3-00F4	243-244	Screen color pointer
00F5-00F6	245-246	Keyboard pointer
00F7-00F8	247-248	RS-232 Rcv pointer
00F9-00FA	249-250	RS-232 Tx pointer
*00FB-00FE	251-254	Operating system free zero page space
00FF	255	Basic storage
0100-010A	256-266	Floating to ASCII work area
0100-013E	256-318	Tape error log
0100-01FF	256-511	Processor stack area
*0200-0258	512-600	Basic input buffer
*0259-0262	601-610	Logical file table
*0263-026C	611-620	Device # table
*026D-0276	621-630	Secondary Address table
*0277-0280	631-640	Keyboard buffer
*0281-0282	641-642	Start of memory for op system
*0283-0284	643-644	Top of memory for op system
0285	645	Serial bus timeout flag
*0286	646	Current color code
0287	647	Color under cursor
*0288	648	Screen memory page
*0289	649	Max size of keyboard buffer
*028A	650	Key repeat (128=repeat all keys)
*028B	651	Repeat speed counter
028C	652	Repeat delay counter
*028D	653	Keyboard Shift/Control flag
028E	654	Last keyboard shift pattern
028F-0290	655-656	Pointer: keyboard decode table
*0291	657	Shift mode switch (0=enabled, 128=locked)
0292	658	Auto scroll down flag(0=on, <>0=off)
0293	659	RS232 control register
0294	660	RS232 command register
0295-0296	661-662	Non standard (Bit time/2-100)

HEX	DECIMAL	DESCRIPTION
0297	663	RS-232 status register
0298	664	Number of bits to send
0299-029A	665-666	Baud rate (full) bit time
029B	667	RS232 receive pointer
029C	668	RS232 input pointer
029D	669	RS232 transmit pointer
029E	670	RS232 output pointer
029F-02A0	671-672	Holds IRQ during tape operations
02A1-02FF	673-767	Program indirects
*0300-0301	768-769	Error message link
0302-0303	770-771	Basic warm start link
0304-0305	772-773	Crunch Basic tokens link
0306-0307	774-775	Print tokens link
0308-0309	776-777	Start new Basic code link
030A-030B	778-779	Get arithmetic element link
030C	780	Storage for 6502 .A register
030D	781	Storage for 6502 .X register
030E	782	Storage for 6502 .Y register
030F	783	Storage for 6502 .P register
0310-0313	784-787	USR jump
0314-0315	788-789	Hardware (IRQ) interrupt vector (FEAB)
0316-0317	790-791	Break interrupt vector (FED2)
0318-0319	792-793	NMI interrupt vector (FEAD)
031A-031B	794-795	OPEN vector (F40A)
031C-031D	796-797	CLOSE vector (F34A)
031E-031F	798-799	Set-input vector (F2C7)
0320-0321	800-801	Set-output vector (F309)
0322-0323	802-803	Restore I/O vector (F3F3)
0324-0325	804-805	INPUT vector (F20E)
0326-0327	806-807	Output vector (F27A)
0328-0329	808-809	Test-STOP vector (F770)
032A-032B	810-811	GET vector (F1F5)
032C-032D	812-813	Abort I/O vector (F3EF)
032E-032F	814-815	user vector (FED2)
0330-0331	816-817	Link to load RAM (F549)
0332-0333	818-819	Link to save RAM (F685)
0334-033B	820-827	??

HEX	DECIMAL	DESCRIPTION
033C-03FB	828-1019	Cassette buffer
0400-07FF	1024-2047	1K Screen memory
(0400-07E7	1024-2023	Video matrix)
(07F8-07FF	2040-2047	Sprite pointers)
0800-9FFF	2048-40959	User Basic area
A000-BFFF	40960-49151	8K BASIC ROM
C000-CFFF	49152-53247	4K RAM
D000-D3FF	53248-54271	6567 Video Chip
D000	53248	Sprite 0 X cmp
D001	53249	Sprite 0 Y cmp
D002	53250	Sprite 1 X cmp
D003	53251	Sprite 1 Y cmp
D004	53252	Sprite 2 X cmp
D005	53253	Sprite 2 Y cmp
D006	53254	Sprite 3 X cmp
D007	53255	Sprite 3 Y cmp
D008	53256	Sprite 4 X cmp
D009	53257	Sprite 4 Y cmp
D00A	53258	Sprite 5 X cmp
D00B	53259	Sprite 5 Y cmp
D00C	53260	Sprite 6 X cmp
D00D	53261	Sprite 6 Y cmp
D00E	53262	Sprite 7 X cmp
D00F	53263	Sprite 7 Y cmp
D010	53264	Sprite X cmp (msb of X coord.)
D011	53265	BIT 7 Raster compare
		BIT 6 Extended color mode
		BIT 5 Bit map mode
		BIT 4 Screen blank
		BIT 3 24/25 Row select (1=25 rows)
		BIT 2-0 Scroll in Y position
D012 (R/O)	53266	Raster read (raster cmp irq write)
D013 (R/O)	53267	Light pen latch X
D014 (R/O)	53268	Light pen latch Y
D015	53269	Sprite enable (1=sprite enabled)
D016	53270	BIT 7-5 Unused
		BIT 4 Multi-color mode
		BIT 3 38/40 Column select (1=40 col.)
		BIT 2-0 Scroll in X position
D017	53271	Sprite expand in "y"
D018	53272	BIT 7-4 Video matrix base
		BIT 3-1 Character space base
D019	53273	BIT 7 Follows IRQ line
		BIT 2 IRQ for sprite to sprite collision
		BIT 1 IRQ for sprite to background collision
		BIT 0 Raster cmp IRQ
D01A	53274	IRQ mask register (0= interrupt disabled)

D01B	53275	Background to sprite priority
D01C	53276	Multi-color sprite select
D01D	53277	Sprite expand in "X"
D01E	53278	Sprite to sprite collision detect
D01F	53279	Sprite to background collision detect

COLOR REGISTERS (BIT 3-0)

D020	53280	Border color
D021	53281	Background color 0
D022	53282	Background color 1
D023	53283	Background color 2
D024	53284	Background color 3
D025	53285	Sprite multi-color register 0
D026	53286	Sprite multi-color register 1
D027	53287	Sprite 0 color
D028	53288	Sprite 1 color
D029	53289	Sprite 2 color
D02A	53290	Sprite 3 color
D02B	53291	Sprite 4 color
D02C	53292	Sprite 5 color
D02D	53293	Sprite 6 color
D02E	53294	Sprite 7 color

D400-D7FF 6581 (SID) SYNTHESIZER CHIP

D400	54272	FREQUENCY LO
D401	54273	FREQUENCY HI
D402	54274	PULSE WIDTH LO
D403	54275	BIT 7-4 UNUSED BIT 3-0 PULSE WIDTH HI
D404	54277	CONTROL REGISTER VOICE 1 BIT 7 NOISE BIT 6 PULSE BIT 5 SAWTOOTH BIT 4 TRIANGLE BIT 3 TEST BIT BIT 2 RING MOD BIT 1 SYNC BIT 0 GATE
D405	54278	ATTACK/DECAY REGISTER
D406	54279	SUSTAIN/RELEASE REGISTER
D407-D40D	54280-54285	CONTROL REGISTERS FOR VOICE 2 (FUNCTIONALLY IDENTICAL TO D400-D406)
D40E-D414	54286-54292	CONTROL REGISTERS FOR VOICE 3 (FUNCTIONALLY IDENTICAL TO D400-D406)

HEX	DECIMAL	DESCRIPTION
0415	54293	CUTOFF FREQUENCY LO
0416	54294	CUTOFF FREQUENCY HI
0417	54295	BIT 7-4 RESONANCE OF FILTER (BIT 3-0 SELECT SIGNALS TO BE ROUTED THROUGH FILTER BITS SET TO ZERO APPEAR DIRECTLY AT AUDIO OUTPUT. BITS SET TO 1 WILL BE PROCESSED THROUGH FILTER) BIT 3 EXTERNAL INPUT BIT 2 VOICE 3 BIT 1 VOICE 2 BIT 0 VOICE 1
0418	54296	(BIT 7-4 SELECT FILTER MODE AND OUTPUT OPTIONS) BIT 7 OFF BIT 6 HIGH PASS BIT 5 BAND PASS BIT 4 LOW PASS BIT 3-0 OUTPUT VOLUME
0419	54297	POT X
041A	54298	POTY
041B	54299	OSCILLATOR 3/RANDOM NUMBER GENERATOR
041C	54230	ENVELOPE 3
D800-DBFF	55296-56319	Color RAM (nibbles)
DC00-DCFF	56320-56575	CIA #1 (Keyboard)
DD00-DDFF	56576-56831	CIA #2 (user port/rs-232) UNFINISHED DOCUMENTATION
DE00-DEFF	56832-57087	I/O expansion block 1
DF00-DFFF	57088-57343	I/O expansion block 2
E000-FFFF	57344-65535	8K Kernal ROM

Bob Yannes
COMMODORE Valley Forge, USA
~~[illegible]~~ 5/19/82

COMMODORE-64 CONNECTOR PIN-OUTS

CONTROL PORT 1
(9-PIN MINI-D)

1- JOYA 0 (Kybd Row 0)
2- JOYA 1 (Kybd Row 1)
3- JOYA 2 (Kybd Row 2)
4- JOYA 3 (Kybd Row 3)
5- POTA Y (Kybd Col 6)
6- BUTTONA/LIGHT PEN (Kybd Row 4)
7- +5 VDC (Total Both Ports 50 mA max)
8- GND
9- POTA X (Kybd Col 6)

CONTROL PORT 2
(9-PIN MINI-D)

1- JOYB 0 (Kybd Col 0)
2- JOYB 1 (Kybd Col 1)
3- JOYB 2 (Kybd Col 2)
4- JOYB 3 (Kybd Col 3)
5- POTB Y (Kybd Col 7)
6- BUTTONB/ (Kybd Col 4)
7- +5 VDC
8- GND
9- POTB X (Kybd Col 7)

SERIAL BUS
(6-PIN DIN)

1- SRQ IN
2- GND
3- ATN IN/OUT
4- CLK IN/OUT
5- DATA IN/OUT
6- RESET

AUDIO/VIDEO
(5-PIN DIN)

1- LUMINANCE OUT
2- GND
3- AUDIO OUT
4- AUDIO IN (3 Vp-p max)
5- COMPOSITE VIDEO

POWER
(7-PIN DIN AS PER CBM SPEC)

1- GND
2- GND
3- GND
4- +5 VDC (Total
5- +5 VDC 1.5 A max)
6- 9 VAC+
7- 9 VAC- (Total 600 mA max)
SHIELD- Earth GND

CASSETTE
(DUAL 6-PIN MALE PC)

1,A- GND
2,B- +5 VDC (Cassette only)
3,C- CASS MTR
4,D- CASS RD
5,E- CASS WRT
6,F- CASS SENSE

USER PORT
(DUAL 12-PIN MALE PC)

1- GND	A- GND
2- +5 VDC (100 mA max)	B- FLAG2
3- RESET	C- PB0
4- CNT1	D- PB1
5- SP1	E- PB2
6- CNT2	F- PB3
7- SP2	H- PB4
8- PC2	J- PB5
9- ATN IN/OUT	K- PB6
10- 9 VAC+ (Total	L- PB7
11- 9 VAC- (100 mA max)	M- PA2
GND	N- GND

CARTRIDGE/EXPANSION
(DUAL 22-PIN FEMALE PC)

1- GND	A- GND
2- +5 VDC (Total USER PORT/	B- ROMH
3- +5 VDC CART 450 mA max)	C- RESET
4- IRQ	D- NMI
5- R/W	E- 02
6- DOT CLOCK	F- A15
7- I/O #1	H- A14
8- GAME	J- A13
9- EXROM	K- A12
10- I/O #2	L- A11
11- ROML	M- A10
12- BA	N- A9
13- DMA	P- A8
14- D7	R- A7
15- D6	S- A6
16- D5	T- A5
17- D4	U- A4
18- D3	V- A3
19- D2	W- A2
20- D1	X- A1
21- D0	Y- A0
22- GND	Z- GND

Commodore Valley Forge, USA
~~5/11/82~~
5/19/82

COMMODORE-64 I/O ASSIGNMENTS

6510 I/O PORT ($0001)

P0- LORAM (Output)
P1- HIRAM (Output)
P2- CHAREN (Output)
> Memory Management

P3- CASS WRT (Output)
P4- CASS SENSE (Input)
P5- CASS MTR (Output)
> Cassette Interface

CIA #1 ($DC00)

PA0- Kybd COL 0/JOYB 0
PA1- Kybd COL 1/JOYB 1
PA2- Kybd COL 2/JOYB 2
PA3- Kybd COL 3/JOYB 3
PA4- Kybd COL 4/BUTTONB
PA5- Kybd COL 5 (Output)
PA6- Kybd COL 6/POTA X,POTA Y SELECT (Output)
PA7- Kybd COL 7/POTB X,POTB Y SELECT (Output)

PB0- Kybd ROW 0/JOYA 0 (Input)
PB1- Kybd ROW 1/JOYA 1 (Input)
PB2- Kybd ROW 2/JOYA 2 (Input)
PB3- Kybd ROW 3/JOYA 3 (Input)
PB4- Kybd ROW 4/BUTTONA/LIGHT PEN (Input)
PB5- Kybd ROW 5 (Input)
PB6- Kybd ROW 6 (Input)
PB7- Kybd ROW 7 (Input)
> Keyboard/Game I/O

FLAG1- CASS RD/SRQ IN
Cassette Interface/SERIAL BUS

SP1- USER
CNT1- USER

CIA #2 ($DD00)

PA0- VA14 (Output)
PA1- VA15 (Output)
VIC High-order Address Bits

PA2- USER/RS-232 TRANSMITTED DATA
RS-232 Interface

PA3- ATN OUT (Output)
PA4- CLK OUT (Output)
PA5- DATA OUT (Output)
PA6- CLK IN (Input)
PA7- DATA IN (Input)
> SERIAL BUS

PB0- USER/RS-232 RECEIVED DATA/Network
PB1- USER/RS-232 REQUEST TO SEND
PB2- USER/RS-232 DATA TERMINAL READY
PB3- USER/RS-232 RING INDICATOR
PB4- USER/RS-232 RECEIVED LINE SIGNAL
PB5- USER
PB6- USER/RS-232 CLEAR TO SEND
PB7- USER/RS-232 DATA SET READY

FLAG2- USER/RS-232 RECEIVED DATA/Network
> USER PORT/RS-232 Interface

SP2- USER/Network Data
CNT2- USER/Network Clock
Network Interface (proposed)

PC2- USER

Bob Yannes
COMMODORE Valley Forge, USA
5/12/82

COMMODORE-64 MEMORY MAP CONTROL

The 6510 Microprocessor used in the COMMODORE-64 can address up to 64K Bytes of memory. As the COMMODORE-64 contains 20K Bytes of ROM and 4K Bytes of I/O in addition to the 64K Bytes of RAM, memory management is provided to allow the memory map to be reconfigured for different applications. Five control lines are provided to select various memory maps. Three of the control lines are provided by the I/O port contained in the 6510 processor, while the remaining two lines are supplied by the plug-in cartridge. The two cartridge lines allow the machine to determine what type of cartridge has been inserted.

The three lines on the 6510 (location 0001) are:

$\overline{\text{LORAM}}$ (Bit 0)- Generally this line can be thought of as a control which banks the 8K Byte BASIC ROM out of the microprocessor's address space (however it interacts with the other control lines to produce various memory configurations). Typically, $\overline{\text{LORAM}}$ is programmed high for normal BASIC operation. If $\overline{\text{LORAM}}$ is programmed low, the BASIC ROM will disappear from the memory map and be replaced by 8K Bytes of RAM from $A000-$BFFF.

$\overline{\text{HIRAM}}$ (Bit 1)- Generally this line can be thought of as a control which banks the 8K Byte KERNAL ROM out of the microprocessor's address space (however this line also interacts with the other control lines to produce various memory configurations). Typically $\overline{\text{HIRAM}}$ is programmed high for normal BASIC operation. If $\overline{\text{HIRAM}}$ is programmed low, the KERNAL ROM will disappear from the memory map and be replaced by 8K Bytes of RAM from $E000-$FFFF.

$\overline{\text{CHAREN}}$ (Bit 2)- This line is used only to bank the 4K Byte Character Generator ROM in or out of the microprocessor's address space. From the microprocessor's point of view, the Character ROM resides in the same address space as the I/O devices (The 4K Byte block from $D000-$DFFF). When $\overline{\text{CHAREN}}$ is programmed high, the I/O devices appear in the processor's address space and the Character ROM is not accessible. If $\overline{\text{CHAREN}}$ is programmed low, the Character ROM will appear in the processor's address space and the I/O devices will not be accessible. For normal BASIC operation, $\overline{\text{CHAREN}}$ is programmed high. Realistically, the microprocessor only needs access to the Character ROM if it is necessary to download the character patterns from ROM into RAM. For most applications, the $\overline{\text{CHAREN}}$ bit will be programmed high. It should be noted that in some cases, $\overline{\text{CHAREN}}$ is overidden by the other control lines ($\overline{\text{CHAREN}}$ will have no effect on any memory configuration which deselects the I/O devices).

The two memory control lines provided by the cartridge are:

$\overline{\text{GAME}}$ (Pin 8)- Generally, this line can be thought of as a control which, when low, causes the COMMODORE-64 memory map to "collapse" into the memory map of the COMMODORE ULTIMAX video game (however this line also interacts with the other control lines to produce various memory configurations). On an ULTIMAX game cartridge, this line would be tied low, indicating to the COMMODORE-64 that a GAME cartridge is being used. In other words, the COMMODORE-64 is downward compatible with the ULTIMAX and any cartridge which runs on the ULTIMAX will also plug directly into the COMMODORE-64 and run. When no cartridge is installed, or when a cartridge other than a game is used, this line is held high.

$\overline{\text{EXROM}}$ (Pin 9)- this line is used to bank the 8K Bytes of RAM from $8000-$9FFF out of the microprocessor's address space and replace it with up to 8K Bytes of ROM in the plug-in cartridge. This line would be tied low on a BASIC "expansion" cartridge such as "VSP" or "Programmer's Aid" indicating to the COMMODORE-64 that a BASIC expansion cartridge is being used. (In order for an expansion cartridge to "autostart", locations $8000-$8001 must contain the Cold Start vector, locations $8002-$8003 must contain the Warm Start vector and locations $8004-$8008 must contain "CBM80" (in ASCII, with the MSB set on each character in "CBM").) When no cartridge is installed, or when a cartridge other than an expansion cartridge is used, this line is held high.

COMMODORE-64 FUNDAMENTAL MEMORY MAP

E000-FFFF	8K <	KERNAL ROM or RAM
D000-DFFF	4K <	I/O or RAM or CHARACTER ROM
C000-CFFF	4K <	RAM
A000-BFFF	8K <	BASIC ROM or RAM or ROM PLUG-IN
8000-9FFF	8K <	RAM or ROM PLUG-IN
4000-7FFF	16K <	RAM
0000-3FFF	16K <	RAM

I/O BREAKDOWN

D000-D3FF	VIC (Video Controller)	1K Bytes
D400-D7FF	SID (Sound Synthesizer)	1K Bytes
D800-DBFF	COLOR RAM	1K nybbles
DC00-DCFF	CIA1 (Keyboard)	256 Bytes
DD00-DDFF	CIA2 (Serial Bus, User Port/RS-232)	256 Bytes
DE00-DEFF	Open I/O slot #1 (CP/M Enable)	256 Bytes
DF00-DFFF	Open I/O slot #2 (Cheap Disk)	256 Bytes

The two open I/O slots are for general purpose user I/O. special purpose /O cartridges (such as IEEE) and have been tentatively designated for nabling the Z-80 cartridge (CP/M option) and for interfacing to a low-st high-speed disk system.

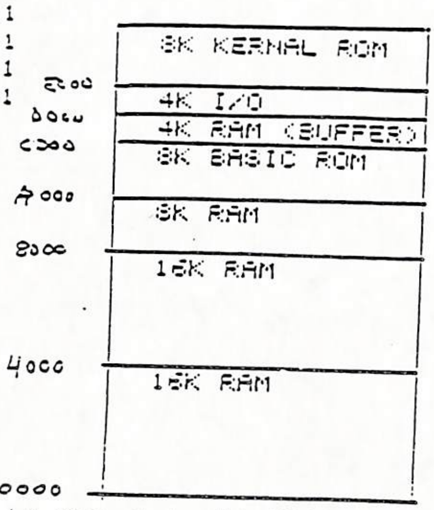
5/19/82

COMMODORE-64 MEMORY MAPS

The following tables list the various memory configurations available on the COMMODORE-64, the states of the control lines which select each memory map and the intended use of each map.

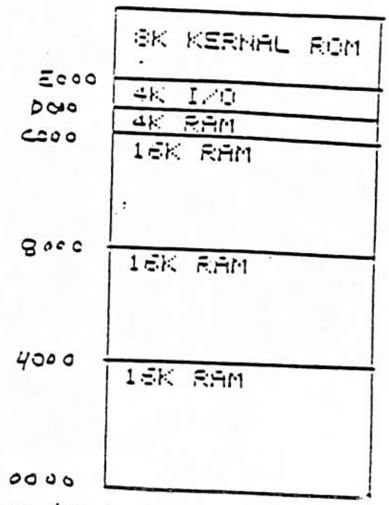
(X=don't care, 0=low, 1=high)

$\overline{\text{LORAM}}$=1
$\overline{\text{HIRAM}}$=1
$\overline{\text{GAME}}$= 1
$\overline{\text{EXROM}}$=1



This is the default BASIC memory map which provides BASIC 2.0 and 38K contiguous bytes of user RAM.

$\overline{\text{LORAM}}$=0
$\overline{\text{HIRAM}}$=1
$\overline{\text{GAME}}$= 1
$\overline{\text{EXROM}}$=X



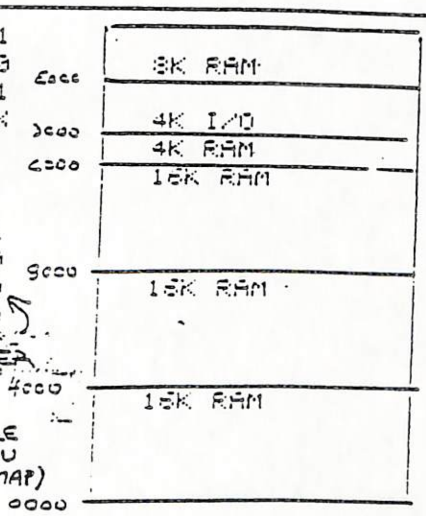
This map is intended for use with softload languages (including CP/M), providing 52K contiguous bytes of user RAM, I/O devices and I/O driver routines.

$\overline{\text{LORAM}}$=1
$\overline{\text{HIRAM}}$=0
$\overline{\text{GAME}}$= 1
$\overline{\text{EXROM}}$=X

or

$\overline{\text{LORAM}}$=1
$\overline{\text{HIRAM}}$=0
$\overline{\text{GAME}}$= 0
$\overline{\text{EXROM}}$=0

(THE CHARACTER ROM IS NOT ACCESSIBLE BY THE CPU IN THIS MAP)

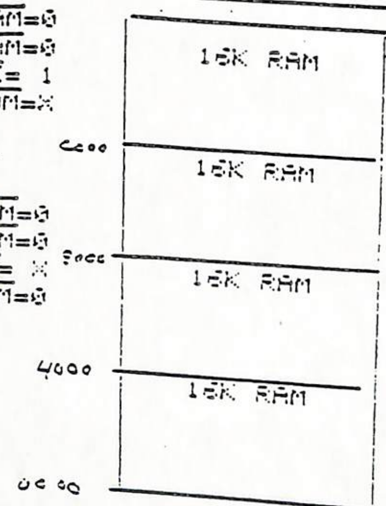


This map provides 60K bytes of RAM and I/O devices. The user must write his own I/O driver routines.

$\overline{\text{LORAM}}$=0
$\overline{\text{HIRAM}}$=0
$\overline{\text{GAME}}$= 1
$\overline{\text{EXROM}}$=X

or

$\overline{\text{LORAM}}$=0
$\overline{\text{HIRAM}}$=0
$\overline{\text{GAME}}$= X
$\overline{\text{EXROM}}$=0



This map gives access to all 64K bytes of RAM. The I/O devices must be banked back into the processor's address space for any I/O operation.

5/19/82

LORAM=1
HIRAM=1
GAME= 1
(ROM=0

Address	Contents
	8K KERNAL ROM
E000	4K I/O
D000	4K RAM (BUFFER)
C000	8K BASIC ROM
A000	8K ROM CARTRIDGE (BASIC EXP)
8000	16K RAM
4000	16K RAM
0000	

This is the standard configuration for a BASIC system with a BASIC expansion ROM. This map provides 32K contiguous Bytes of user RAM and up to 8K Bytes of BASIC "Enhancement".

LORAM=1
HIRAM=1
GAME= 0
EXROM=0

Address	Contents
	8K KERNAL ROM
E000	4K I/O
D000	4K RAM (BUFFER)
C000	16K ROM CARTRIDGE
8000	16K RAM
4000	16K RAM
0000	

This map provides 32K contiguous Bytes of user RAM and up to 16K Bytes of plug-in ROM for special ROM-based applications which don't require BASIC (word processors, other languages, etc.).

LORAM=0
HIRAM=1
GAME= 0
EXROM=0

Address	Contents
	8K KERNAL ROM
E000	4K I/O
D000	4K RAM (BUFFER)
C000	8K ROM CARTRIDGE
A000	8K RAM
8000	16K RAM
4000	16K RAM
0000	

This map provides 40K contiguous Bytes of user RAM and up to 8K Bytes of plug-in ROM for special ROM-based applications which don't require BASIC.

LORAM=X
HIRAM=X
GAME= 0
EXROM=1

Address	Contents
	8K CARTRIDGE ROM
E000	4K I/O
D000	4K OPEN
C000	8K OPEN
A000	8K CARTRIDGE ROM
8000	16K OPEN
4000	12K OPEN
1000	4K RAM
0000	

This is the ULTIMAX video game memory map. Note that the 2K Byte "expansion RAM" for the ULTIMAX, if required, is accessed out of the COMMODORE-64 and any RAM in the cartridge is Ignored.

MISCELLANEOUS INFO

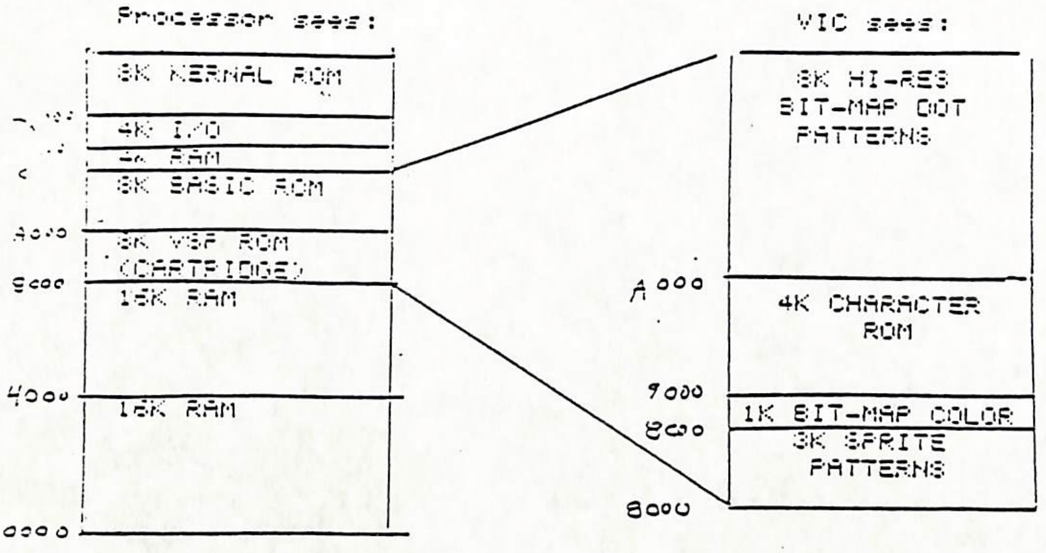
~~[illegible]~~
5/19/82

1.) The $\overline{CHAREN}$ control line will bank the Character ROM into the processor's address space in place of the I/O devices with any memory map that includes I/O devices (with the exception of $\overline{LORAM}$=1, $\overline{HIRAM}$=0, $\overline{GAME}$=0 and $\overline{EXROM}$=0). Those memory configurations which replace the I/O with RAM (giving access to all 64K Bytes of RAM) are unaffected by the CHAREN line.

2.) In any memory map containing ROMs, a WRITE to a ROM location will store data in the RAM which the ROM overlays. That is, writing to a ROM location will, in fact, store data in the "hidden" RAM. This allows, for example, a HI-RES screen to reside underneath a ROM and be changed by the processor without banking the RAM into the processor's address space. Naturally, a R from a ROM location will return the contents of the ROM and not the "hidden RAM.

3.) The ULTIMAX memory map configuration matches the memory map of the COMMODORE ULTIMAX video game exactly. All open memory blocks in the ULTIMAX are also open on the COMMODORE-64 when this memory map is selected, therefore, any cartridge or device that plugs into the ULTIMAX will also work on the COMMODORE-64. Note that there are only 2K Bytes of RAM in the ULTIMAX and for applications requiring more RAM, the cartridge can contain up to 2K Bytes of expansion RAM. If an ULTIMAX cartridge containing expansion RAM is installed in the COMMODORE-64, the RAM in the cartridge will be ignored and the COMMODORE-64 will use its own internal RAM instead. The COMMODORE-64 Character ROM is not available to ULTIMAX cartridges. As on the ULTIMAX, the 4K Bytes of cartridge ROM from \$F000-\$FFFF are accessib by the VIC video chip and should contain all character and Sprite patterns.

4.) All of the memory maps described apply ONLY TO THE MICROPROCESSOR. The VIC video display chip sees a much simpler memory map, regardless of the state of the memory control lines (LORAM, etc.). The VIC can ONLY see RAM or the Character ROM. The VIC chip itself is capable of addressing only 16K Bytes of memory. In order to expand this address space, two I/O Port lines, $\overline{VA14}$ and $\overline{VA15}$ (location \$DD00, bits 0 and 1), are used to form the most significant bits of the VIC address. Essentially these two bits select which of the four 16K banks of memory the VIC can look at. Note the these port bits are INVERTED, that is, setting $\overline{VA14}$ and $\overline{VA15}$ HIGH will cause the VIC to access the LOW 16K block of memory (from \$0000-\$3FFF). Setting $\overline{VA14}$ low and $\overline{VA15}$ high will select the second 16K block (\$4000-\$7FFF), etc. The Character ROM is only available to the VIC chip in Bank 0 (\$0000-\$3FFF) and Bank 2 (\$8000-\$BFFF) and appears in the ~~UPPER~~ MIDDLE 4K Bytes of these two banks. In order for the VIC chip to access the Character ROM, bits CB13 and CB12 in the CHARACTER BASE REGISTER of the VIC (Register \$18) must be set to a particular state. CB13 should be set low (0) and CB12 sho be set high (1). Note that the VIC will not be able to access the 4K Bytes of RAM which the Character ROM displaces. For those applications in which the VIC chip must access a full 16K Bytes of RAM, or for some reason must access RAM in this 4K block of the desired bank, Bank 1 (\$4000-\$7FFF) or Bank 3 (\$C000-\$FFFF) should be used, as the Character ROM is disabled in these banks.



The VSP package will automatically handle the bank switching necessary to plot into the RAM underneath the BASIC and VSP ROMs.

6.) Explanation of certain cartridge lines:

Pin 6, DOT CLOCK- This line is the 8.18 MHz video dot clock from which all system timing is derived.

Pin 7, $\overline{\text{I/O \#1}}$- This is a negative active chip select for the block of memory from $0E00-$0EFF (256 Bytes).

Pin 10, $\overline{\text{I/O \#2}}$- This is a negative active chip select for the block of memory from $0F00-$0FFF (256 Bytes).

Pin 11, $\overline{\text{ROML}}$- This is a negative active chip select for the block of memory from $8000-$9FFF (8K Bytes). This is true for all COMMODORE-64 memory maps, including ULTIMAX. Any cartridge ROMs which <u>are</u> to appear in the $8000 block (VSP, etc.) should have their CS pin connected to this line.

Pin 12, BA- This is the BUS AVAILABLE signal from the VIC chip. This line will go low three cycles before the VIC takes over the system busses and remain low until VIC is finished fetching display information.

Pin 13, $\overline{\text{DMA}}$- This line allows external devices, such as the Z-80 microprocessor cartridge, to take control of the system busses. When pulled low, the address bus, data bus and R/W line of the 6510 processor are Tri-stated. This line should only be pulled low when the 02 clock is low. In addition, the VIC chip will continue to perform display DMA, therefore the external device must conform to the VIC timing. This line is pulled-up on the COMMODORE-64.

Pin 8, $\overline{\text{ROMH}}$- This is a negative active chip select for the block of memory from $A000-$BFFF (8K Bytes) for all COMMODORE-64 memory maps except for the ULTIMAX map. For the ULTIMAX map, this line is a negative active chip select for the block of memory from $E000-$FFFF (8K Bytes). Any cartridge ROMs which are to appear in the $A000 block (in place of BASIC) should have their CS pin tied to this line. Any ULTIMAX game ROM which is to appear in the $E000 block should have the $\overline{\text{CS}}$ pin tied to this line.

Book 1Ø3

SOFTWARE APPLICATION NOTE 1001

Authors: Joe McEnerney, Eric Cotton, and Bill Hindorff

Subject: Sprite Movement

Television Standard: This Application Note explains sprite movement for both PAL and NTSC versions of the VIC II chip.

I. X MOVEMENT and X COMPARE

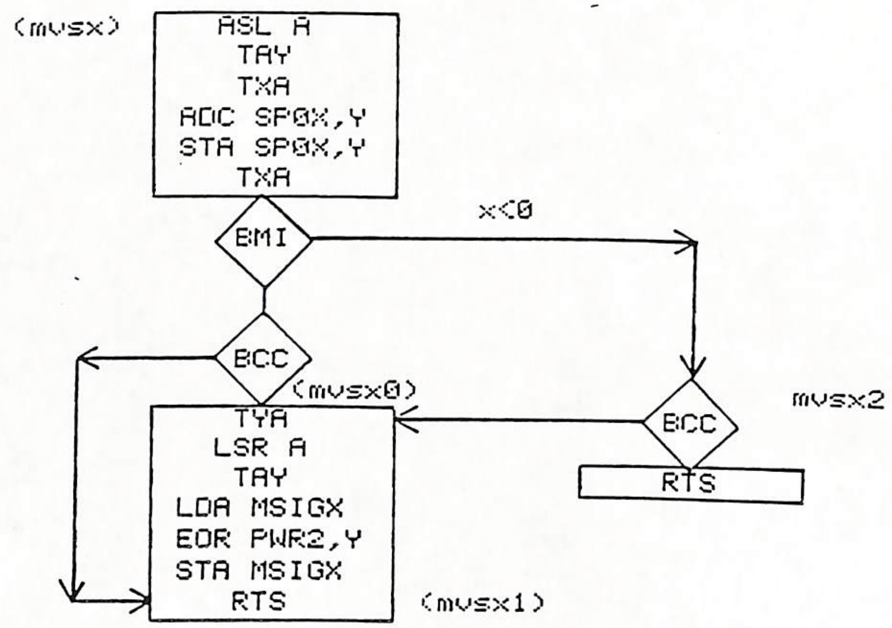
A. X MOVEMENT

1. Abstract: There are \$200 possible x positions for a sprite on a NTSC version of the VIC II chip (\$0 <= x <= \$1FF), while on the PAL version there are \$1F8 (\$0 <= x <= \$1F7). Because positions \$1F8 through \$1FF do not exist on the PAL version, two separate x movement subroutines are presented below. The first, MVSX, is for sprites expanded or unexpanded in x on NTSC systems, and/or for sprites **unexpanded** in x on PAL systems. The second, UNIMVX, is a universal routine: It works for all sprites, expanded or unexpanded in x, on both television standards. However, because it is slower and uses more memory than MVSX, the programmer should limit its use to those occasions when his program requires movement of sprites expanded in x on PAL systems across the range \$1F8 <= x <= \$1FF.

Immediately before calling either of the subroutines the accumulator must be loaded with the sprite number (0 through 7) and the x register with the two's complement offset. The user should note the additional requirements of UNIMVX. In both, however, it does not matter whether the sprite is expanded or unexpanded in y.

2. Exposition - MVSX: The following deals with x movement of sprites expanded or unexpanded in x on NTSC television systems, and/or sprites unexpanded in x on PAL systems.

a. Flow chart:



b. Program listings:

1) Source:

```
1000 .PAGE 'MVSX'
1010 ;
1020 ;CONSTANTS
1030 SP0X    =$D000          ;START OF SPRITE X,Y POSITIONS
1040 MSIGX   =$D010          ;VIC REGISTER CONTAINING X MSB'S
1050 ;
1060 *=$2000
1070 ;
1080 MVSX     ASL A          ;.Y=2*SPR#
1090          TAY
1100          TXA            ;.A=2'S COMPLEMENT OFFSET
1110          ADC SP0X,Y
1120          STA SP0X,Y
1130          TXA
1140          BMI MVSX2      ;2'S COMPLEMENT OFFSET<0?
1150          BCC MVSX1      ;NO CARRY FROM ADC ABOVE...SO EXIT
1160 MVSX0    TYA            ;CORRECT THE MSB OF SPRITE X-COORD
1170          LSR A          ;.A=INT(.A/2)
1180          TAY            ;.Y=.A
1190          LDA MSIGX      ;MSIGX IS VIC REGISTER OF X MSB'S
1200          EOR PWR2,Y     ;PWR2 IS A TABLE OF THE POWERS OF TWO
1210          STA MSIGX
1220 MVSX1    RTS
1230 ;
1240 MVSX2    BCC MVSX0
1250          RTS
1260 ;
1270 PWR2     .BYTE $01,$02,$04,$08,$10,$20,$40,$80
1280 .END
```

2) Hex dump (as assembled at $2000):

```
.:2000  0A A8 8A 79 00 D0 99 00 D0 8A
.:200A  30 0F 90 0C 98 4A A8 AD 10 D0
.:2014  59 1E 20 8D 10 D0 60 90 F1 60
.:201E  01 02 04 08 10 20 40 80
```

3) Data statements (as assembled at $2000):

```
1000 data 10,168,138,121,0,208,153,0,208,138
1010 data 48,15,144,12,152,74,168,173,16,208
1020 data 89,30,32,141,16,208,96,144,241,96
1030 data 1,2,4,8,16,32,64,128
```

c. Memory/Register requirements: This routine requires 38 ($26) bytes of memory, including 8 bytes for the table of the powers-of-two (PWR2). It uses the accumulator and the x and y registers.

d. Worst case execution time is 55 ($37) cycles (53.90 micro-seconds).

2. Exposition - UNIMVX: This section deals with x movement of sprites expanded or unexpanded in x on NTSC or PAL television systems. UNIMVX operates the same as MVSX insofar as the accumulator must first be loaded with the sprite number (0 through 7) and the y register with the two's complement offset. However, because this is a universal (NTSC or PAL) x movement routine, there must also be a routine to check on which television system the routine is being used and then communicate this information to the UNIMVX subroutine. Because the NTSC system is based on 262 raster lines per screen while the PAL system is based on 312, the presence of a raster 263 (or greater) would imply the PAL system. Thus we include TVSTD which checks to see if a raster line greater than 264 is scanned. (Raster 264 is used as the compare value instead of 263 to insure that the correct system is chosen.) If the raster passes 264, then the PAL system is in use; otherwise, the NTSC system is. To communicate its finding to UNIMVX, TVSTD stores the high and low bytes of of the first illegal x position for the system in the variables MODH and MODL, respectively: $1F8 for the PAL and $200 for the NTSC.

Following UNIMVX there are two additional subroutines, VTAMX and ATVMX. VTAMX need only be executed once, sometime before UNIMVX is called the first time. Its purpose is to copy the x position MSB's from MSIGX ($D010) to individual locations in RAM (starting with SMSB). The second routine, ATVMX, does the opposite: It copies the RAM MSB's back into MSIGX. It should be executed after all of the sprites' x positions have been updated.

In summary, the user must follow this procedure: Execute TVSTD and VTAMX once each at the begining of the program. Then, every time a sprite is to be moved in the x direction: 1)Load the accumulator with the sprite number. 2)Load the x register with the two's complement offset. 3)Execute UNIMVX. 4)Loop back to step 2 if more sprites are to be updated. 5)Execute VTAMX. (See Section III. B. of

this Application Note for an example using UNIMVX.)

b. Source listing:

```
1000 .PAG 'UNIMVX'
1010 ;
1020 ;VARIABLES:
1030 *=$0010                  ;OR SOMEWHERE IN PG 0
1040 OSH    *=*+1             ;OFFSET HIGH
1050 OSL    *=*+1             ;OFFSET LOW
1060 MODH   *=*+1             ;MODULUS HIGH
1070 MODL   *=*+1             ;MODULUS LOW
1080 TX     *=*+1             ;TEMP X
1090 TA     *=*+1             ;TEMP A
1100 SMSB   *=*+8             ;AUXILIARY SPRITE MSB BYTES
1110 ;
1120 ;CONSTANTS:
1130 VIC    =$D000            ;START OF VIC CHIP
1140 SP0X   =$D000            ;SPRITE 0 X-COORD LSB
1150 MSIGX  =$D010            ;SPRITE X-COORD MOST SIG BITS
1160 RASTER=$D012             ;RASTER LSB
1170 ;
1180 *=$2000
1190 ;
1200 ;
1210 ;              TELEVISION STANDARD CHECK
1220 ; SETS UP HIGH AND LOW BYTES OF MODULUS (MODH,MODL)
1230 ;          ACCORDING TO T.V. STANDARD IN USE:
1240 ;      NTSC: $200                  PAL: $1F8
1250 ;
1260 TVSTD  SEI               ;DISABLE IRQS
1270        LDX #$01          ;SET UP PAL MODULUS MSB
1280        LDY #$F8          ;SET UP PAL MODULUS LSB
1290 TVS0   LDA VIC+$11       ;LOOK AT RASTER MSB. IS IT A 1 ?
1300        BPL TVS0          ;IF 0 THEN RASTER < 256. SO LOOK AGAIN
1310 TVS1   LDA #$08          ;RASTER > 255 ...BUT...
1320        CMP RASTER        ;IS IT GREATER THAN 264 ?
1330        BCC TVS2          ;IF YES THEN BRANCH. TV STD = PAL !!
1340        LDA VIC+$11       ;IF NO THEN CHECK FOR MSB OF RASTER = 1
1350        BMI TVS1          ;IF YES THEN GOTO TVS1
1360        LDX #$02          ;SET UP NTSC MODULUS MSB. TV STD = NTSC
1370        LDY #$00          ;SET UP NTSC MODULUS LSB
1380 TVS2   STX MODH          ;STORE IN MODULUS FOR FUTURE USE
1390        STY MODL
1400        CLI
1410        RTS
1420 ;
1430 ;
1440 ;
1450 ;          UNIVERSAL MOVE SPRITE IN X DIRECTIONS
1460 ;
1470 ; A REG = SPRITE NO.    X REG = OFFSET (2'S COMPLEMENT FORM)
1480 ;
1490 UNIMVX STX TX            ;PROTECT X
1500        STX OSL           ;SET UP OFFSET LOW
```

```
1510          TAX                 ;A=X
1520          ASL A               ;A=2*A
1530          TAY                 ;Y=A
1540          LDA #0              ;CLEAR OFFSET HIGH
1550          STA OSH
1560          CLC                 ;CLEAR CARRY FOR LATER
1570          LDA OSL             ;CHECK FOR NEGATIVE OFFSET
1580          BPL UMX0            ;IF POSITIVE THEN BRANCH
1590          EOR #$FF            ;PERFORM 2'S COMPLEMENT OPERATION
1600          ADC #1
1610          STA OSL             ;PUT RESULTS IN OSL AND THEN
1620          SEC                 ;FORM THE MODULAR COMPLEMENT OF
1630          LDA MODL            ;THE OFFSET BY SUBTRACTING IT
1640          SBC OSL             ;FROM THE MODULUS
1650          STA OSL
1660          LDA MODH            ;PAL 504 , NTSC 512
1670          SBC OSH
1680          STA OSH
1690          CLC
1700 UMX0     LDA SP0X,Y          ;ADD OFFSET TO SPRITE X
1710          ADC OSL             ;[SMSB+X,SP0X+Y]=...
1720          STA SP0X,Y          ;   ...[SMSB+X,SP0X+Y]+[OSH,OSL]
1730          LDA SMSB,X
1740          ADC OSH
1750          STA SMSB,X
1760          SEC                 ;IS THE SUM >= MODULUS?
1770          LDA SP0X,Y          ;CHECK BY SUBTRACTING
1780          SBC MODL
1790          STA TA              ;CATCH FOR LATER USE
1800          LDA SMSB,X
1810          SBC MODH
1820          BCC UMX1            ;IF SUM < MOD THEN EXIT
1830          STA SMSB,X          ;OTHERWISE CORRECT X-COORD
1840          LDA TA
1850          STA SP0X,Y
1860 UMX1     TYA                 ;RESTORE A REG
1870          LSR A
1880          LDX TX              ;RESTORE X REG
1890          RTS
1900 ;
1910 ;          TRANSFER SPRITE MSB BYTES TO MSIGX BITS
1920 ;
1930 ATVMX    LDX #7
1940 ATV0     LDA SMSB,X
1950          LSR A
1960          ROL MSIGX
1970          DEX
1980          BPL ATV0
1990          RTS
2000 ;
2010 ;          TRANSFER SPRITE MSIGX BITS TO MSB BYTES
2020 ;
2030 VTAMX    LDX #7
2040          LDA MSIGX
2050 VTA0     LSR SMSB,X
2060          ASL A
```

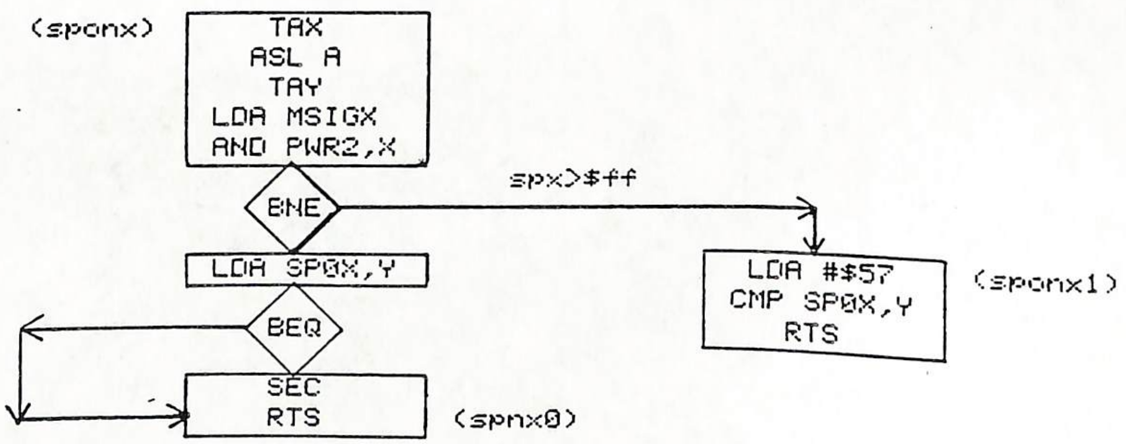
```
2070          ROL SMSB,X
2080          DEX
2090          BPL VTA0
2100          RTS
2110 ;
2120 ;
2130 .END
```

B. X COMPARE

1.Abstract: This routine checks to see if any bit of a given sprite is in the viewable region of the screen. The accumulator must be loaded with the sprite number (0 through 7) prior to use. At RTS time the carry flag indicates the answer. If carry=1, then the sprite is on the screen; if carry=0, then the sprite is off screen.

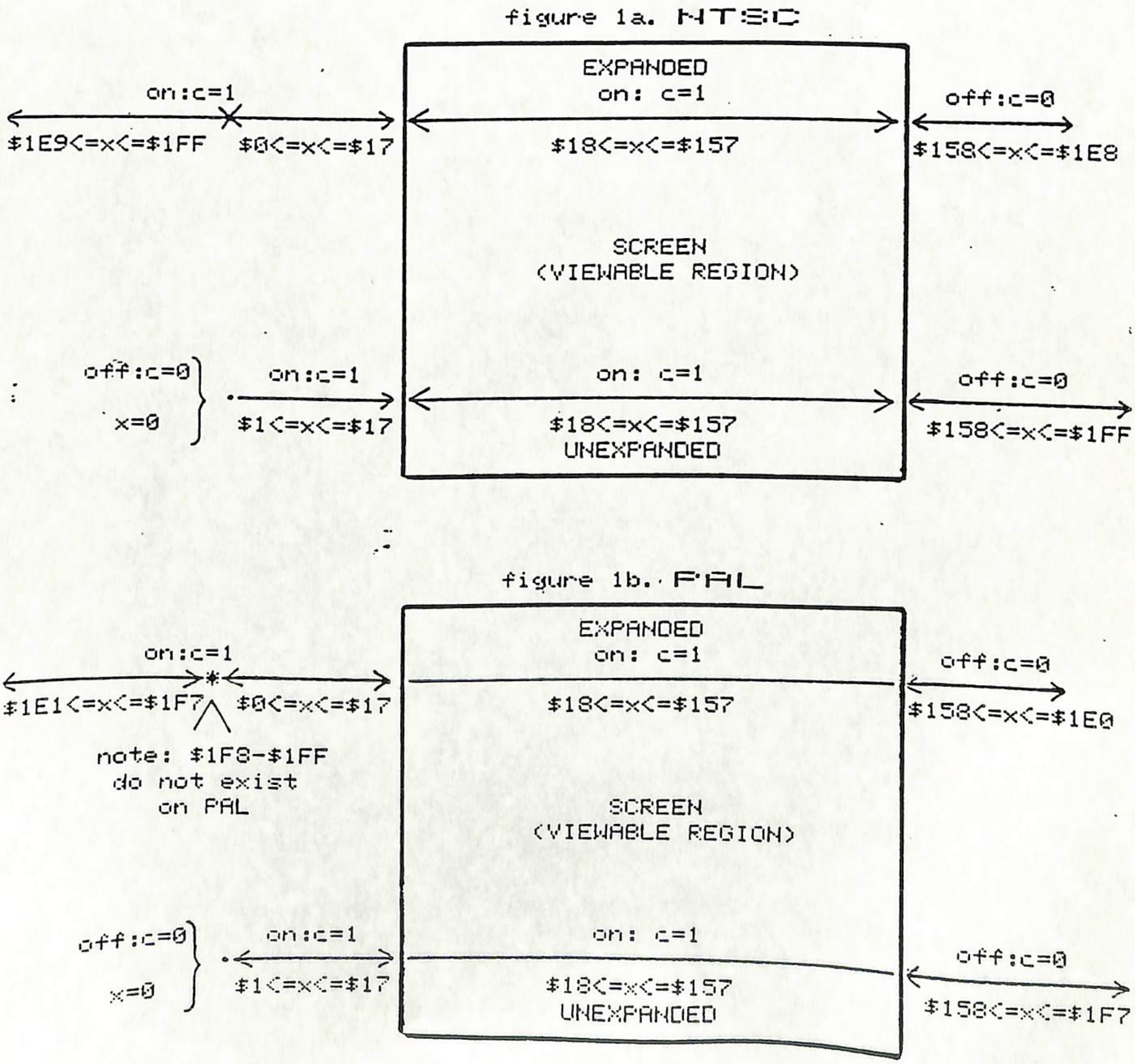
2. Exposition: The following exposition deals with x 40 column window compare of sprites unexpanded in x on NTSC or PAL television systems. Section I. B. 3. explains how this routine may be modified for sprites expanded in x.

a. Flow chart



b. Diagrams: The following diagrams illustrate the on and off screen areas for a sprite. Figure 1a is for NTSC systems and figure 1b is for PAL. In each, the top half shows the case for sprites expanded in x, while the bottom shows the unexpanded case. If the SPONX (or SONXX for exp. sprites) is called with the sprite in the on-screen area, carry would be set upon returning. If called with the sprite is entirely in the off-screen area, carry would be clear.

figure1. X COMPARE



c. Program listing as assembled at $2000:

1) Source:

```
1000 .PAGE 'SPONX'           ;UNEXPANDED
1010 ;
1020 ;CONSTANTS
1030 SP0X    =$D000          ;START OF VIC X,Y POSITIONS
1040 MSIGX   =$D010          ;VIC REGISTER CONTAINING X MSB'S
1050 ;
1060 ;
1070 *=$2000
1080 ;
1090 SPONX   TAX             ;AT START .A SHOULD CONTAIN SPR#
1100         ASL A           ;THE CARRY FLAG IS CLEARED
1110         TAY             ;.Y=2*SPR#
1120         LDA MSIGX       ;MSIGX IS VIC REGISTER OF X MSB'S
1130         AND PWR2,X      ;PWR2 IS A TABLE OF POWERS OF 2
1140         BNE SPONX1      ;IF X>$FF THEN CMP LOW ORDER PART
1150         LDA SP0X,Y      ;SEE IF 8 LSB'S OF X ARE ZEROS
1160         BEQ SPONX0      ;IF X=0 THEN EXIT
1170         SEC             ;  ELSE IF 0<X<$FF THEN SET CARRY
1180 SPONX0  RTS             ;EXIT
1190 ;
1200 SPONX1  LDA #$57        ;X>$157? ($157 IS LAST X ON SCREEN)
1210         CMP SP0X,Y      ;THE CARRY FLAG IS UPDATED
1220         RTS
1230 ;
1240 PWR2    .BYTE $01,$02,$04,$08,$10,$20,$40,$80
1245 ;
1250 .END
```

2) Hex dump (as assembled at $2000):

```
.:2000  AA 0A A8 AD 10 D0 3D 18 20 D0
.:200A  07 B9 00 D0 F0 01 38 60 A9 57
.:2014  D9 00 D0 60 01 02 04 08 10 20
.:201E  40 80
```

3) Data statements (as assembled at $2000):

```
1000 data 170,10,168,173,16,208,61,24,32,208
1010 data 7,185,0,208,240,1,56,96,169,87
1020 data 217,0,208,96,1,2,4,8,16,32
1030 data 64,128
```

d. Memory/Register requirements: This routine requires 32 ($20) bytes of memory, including 8 bytes for the table of powers-of two (PWR2). It uses the accumulator and the x and y registers.

e. Worst case execution time is 31 ($1F) cycles (30.38 micro-seconds).

3. Modification for Expanded Sprites: This routine must be

modified if it is to be used with a sprite expanded in x. The following routine, SONXX, is SPONX rewritten for use with sprites expanded in x on NTSC television systems. Its use is identical to that of SPONX. To use this with a PAL VIC chip, change line 1180 from CMP #$E9 to CMP #$E1.

```
1000 .PAGE 'SONXX'              ;EXPANDED
1010 ;
1020 ;CONSTANTS
1030 SP0X    =$D000             ;START OF VIC X,Y POSITIONS
1040 MSIGX   =$D010             ;VIC REGISTER CONTAINING X MSB'S
1050 ;
1060 ;
1070 *=$2000
1080 ;
1090 SONXX  TAX                 ;AT START .A SHOULD CONTAIN SPR#
1100        ASL A               ;THE CARRY FLAG IS CLEARED
1110        TAY                 ;.Y=2*SPR#
1120        LDA MSIGX           ;MSIGX IS VIC REGISTER OF X MSB'S
1130        AND PWR2,X          ;PWR2 IS A TABLE OF POWERS OF 2
1140        BEQ SONXX0          ;IF X<$FF THEN SPR ON SCREEN
1150        LDA SP0X,Y          ;   ELSE IF X<$158
1160        CMP #$58            ;   THEN SPR ON SCREEN
1170        BCC SONXX0
1180        CMP #$E9            ;OTHERWISE, IS X=>$1E9?
1190        RTS                 ;EXIT
1200 ;
1210 SONXX0 SEC                 ;X>$157? ($157 IS LAST X ON SCREEN)
1220        RTS                 ;EXIT
1230 ;
1240 PWR2    .BYTE $01,$02,$04,$08,$10,$20,$40,$80
1250 ;
1260 .END
```

a. Hex dump (as assembled at $2000):

```
.:2000  AA 0A A8 AD 10 D0 3D 17 20 F0
.:200A  0A B9 00 D0 C9 58 90 03 C9 E9
.:2014  60 38 60 01 02 04 08 10 20 40
.:201E  80
```

b. Data statements (as assembled at $2000):

```
1000 data 170,10,168,173,16,208,61,23,32,240
1010 data 10,185,0,208,201,88,144,3,201,233
1020 data 96,56,96,1,2,4,8,16,32,64
1030 data 128
```

4. NOTES:

a. The changes below modify SPONX to set carry only if the entire sprite is on screen but clear carry otherwise. (Line numbers refer to source listing.)

1) Sprites unexpanded in x:
   a) change line 1160 from BEQ SPONX0 to CMP #$18
   b) delete line1170
   c) change line 1200 from LDA #$57 to LDA #$40

2) Sprites expanded in x:
   a) change line 1160 from BEQ SPONX0 to CMP #$18
   b) delete line1170
   c) change line 1200 from LDA #$57 to LDA #$28

b. Sometimes it is more useful to know if any part of a sprite is entirely on screen but not at an edge, such as the case when a sprite is to be kept confined to the screen. The changes below modify SPONX in this regard: carry is cleared if the sprite is all or partially offscreen or if it is at either the left or right edge. Care should be taken when the movement offset is greater than 1. The sprite could jump over an edge and flag the user only after the sprite is all or partially off screen. If the offset is 1, however, this will not happen. (Again, line numbers refer to the source listing.)

1) Sprites unexpanded in x:
   a) change line 1160 from BEQ SPONX0 to CMP #$19
   b) delete line1170
   c) change line 1200 from LDA #$57 to LDA #$3F

2) Sprites expanded in x:
   a) change line 1160 from BEQ SPONX0 to CMP #$19
   b) delete line1170
   c) change line 1200 from LDA #$57 to LDA #$27

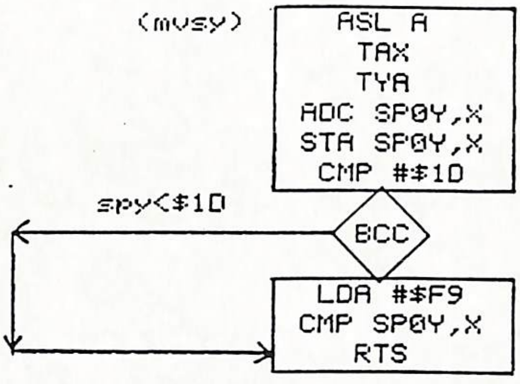
II. Y MOVEMENT and Y COMPARE

A. Y MOVEMENT
together in one routine
B. Y COMPARE

1. Abstract: This routine moves a given sprite in the y axis by an offset specified by a two's complement amount loaded into the Y register prior to calling this routine. The sprite number (0 through7) must be loaded into the accumulator before the routine is executed. At RTS time the carry flag reflects the status of the sprite with respect to the exterior region. Thus, if carry=1 then the sprite is on the screen; if carry=0 then the sprite is off screen. The subroutine below works on both NTSC and PAL television systems. Only slight modification is necessary to make the routine work with sprites expanded in y.

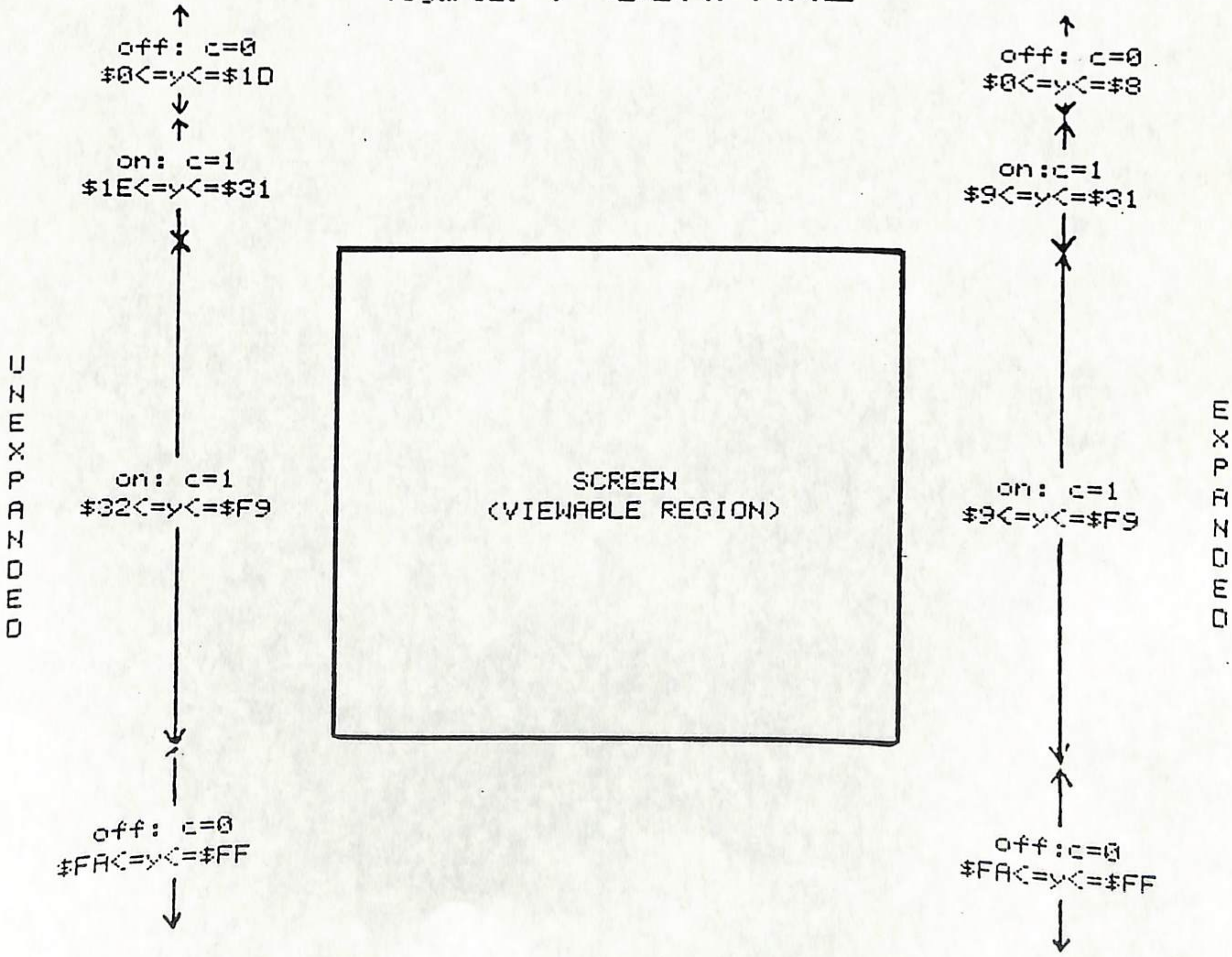
2. Exposition:

a. Flow chart:



b. Diagram: Figure 2 below shows the ranges of sprite y positions for which the sprite is on-screen (carry=1) or off-screen (carry=0).

figure2. 'Y' COMPARE



c. Program listings:

1) Source:

```
1000 .PAGE 'MVSY'
1010 ;
1020 ;CONSTANT
1030 SP0Y   =$D001          ;START OF SPRITE X,Y POSITIONS
1040 ;
1050 *=$2000
1060 ;
1070 MVSY   ASL A           ;AT START .A SHOULD BE LOADED WITH SPR#
1080        TAX             ;.X=2*SPR#
1090        TYA             ;OFFSET MOVED INTO .A
1100        ADC SP0Y,X      ;OFFSET IS ADDED TO SPR Y POSITION
1110        STA SP0Y,X      ;SUM IS NEW Y POSITION
1120        CMP #$1D        ;IF Y<$1D THEN C=0
1130        BCC MVSY0       ;  (AND BRANCH TO EXIT)
1140        LDA #$F9        ;  ELSE IS Y>$F9? (LAST Y ON SCREEN)
1150        CMP SP0Y,X      ;CARRY IS UPDATED ACCORDINGLY
1160 MVSY0  RTS             ;EXIT
1170 ;
1180 .END
```

2) Hex dump (as assembled at $2000):

```
.:2000  0A AA 98 7D 01 D0 9D 01 D0 C9
.:200A  1D 90 05 A9 F9 DD 01 D0 60
```

3) Data statements (as assembled at $2000):

```
1000 data 10,170,152,125,1,208,157,1,208,201
1010 data 29,144,5,169,249,221,1,208,96
```

d. Memory/Register requirements: This routine requires 19 ($13) bytes of memory as well as the accumulator and the x and y registers.

e. Worst case execution time is 31 ($1F) machine cycles (30.38 micro-seconds).

f. Limitations: This routine assumes that the VIC II chip is in 25 row mode and that the sprite in question is unexpanded in y (unless the routine has been changed as specified in section g below).

g. If this routine is to be used with sprites expanded in y, then change line 1120 of the editor listing from CMP #$1D to CMP #$0E. This change is applicable when using y-expanded sprites on either NTSC or PAL television systems.

3. NOTES:

a. The changes below modify MVSY to set carry only if the entire sprite is on screen, but clear it otherwise. (Line numbers refer to the source listing.)

1) Sprites unexpanded in y:
   a) change line 1120 from CMP #$1D to CMP #$32
   c) change line 1140 from LDA #$F9 to LDA #$E5

1) Sprites expanded in y:
   a) change line 1120 from CMP #$1D to CMP #$32
   c) change line 1140 from LDA #$F9 to LDA #$D0

b. Sometimes it is more useful to know if any part of a sprite is entirely on screen but not at an edge, such as the case when a sprite is to be kept confined to the screen. The changes below modify MVSY in this regard: carry is cleared if the sprite is all or partially offscreen or if it is at either the top or bottom edge. Care should be taken if the movement offset is greater than 1. The sprite could jump over an edge and flag the user only after the sprite is all or partially off screen. If the offset is 1, however, this will not happen. (Again, line numbers refer to the source listing.)

1) Sprites unexpanded in y:
   a) change line 1120 from CMP #$1D to CMP #$33
   c) change line 1140 from LDA #$F9 to LDA #$E4

1) Sprites expanded in y:
   a) change line 1120 from CMP #$1D to CMP #$34
   c) change line 1140 from LDA #$F9 to LDA #$Cf

III. EXAMPLES

A. EXAMPLE 1

The example below uses all three main subroutines presented in this Application Note. Its purpose is to demonstrate the use of the routines in an actual program. When executed, this program will bounce an unexpanded sprite around the screen on either television system. The MVSX and MVSY subroutines move the sprite until it is entirely off the screen (as indicated by SPONX or MVSY). If, after moving the sprite in the x direction, the x compare indicates that the sprite has gone off screen, then the x direction is reversed. The same is true for y movement.

By modifying the subroutines as explained elsewhere in this note, the example can be made to work with an expanded sprite. (Remove lines 1250 and 1260 of the example if an expanded sprite is to be used).

NOTE:

The subroutine WAIT has been has been added to slow down the example and thus better enable the programmer to follow its execution. This delay loop lasts for about 20662 machine cycles, equivalent to about 20.25 milliseconds.

```
1000 .PAGE 'EXAMPLE1'
1010 ;
1020 *=$0002
1030 SW1     *=*+1              ;SWITCH 1: $0=RIGHT, $FF=LEFT
1040 SW2     *=*+1              ;SWITCH 2: $0=DOWN, $FF=UP
1050 ;
1060 ;CONSTANTS
1070 SP0X    =$D000             ;SPRITE #0 LSB'S OF X POSITION
1080 SP0Y    =$D001             ;SPR#0 Y POSITION
1090 MSIGX   =$D010             ;VIC REGISTER CONTAINING X MSB'S
1100 SPENA   =$D015             ;SPRITE ENABLE REGISTER
1110 YXPAND  =$D017             ;Y-EXPANSION REGISTER
1120 XXPAND  =$D01D             ;X EXPANSION REGISTER
1130 ;
1140 ;
1150 *=$2000
1160 ;
1170 EXAMP   LDA #$18           ;INITIALIZE SPRITE 0
1180         STA SP0X           ;INITIALIZE X POSITION=$18
1190         LDA #$32
1200         STA SP0Y           ;INITIALIZE Y POSITION=$32
1210         LDA #$01
1220         STA SPENA          ;ENABLE SPRITE 0
1230         LDA #$00
1240         STA MSIGX          ;SET X MSB TO 0
1250         STA XXPAND         ;NO X EXPANSION
1260         STA YXPAND         ;NO Y EXPANSION
1270         STA SW1            ;INITIALIZE TO "RIGHT"
1280         STA SW2            ;  AND "DOWN"
1290 ;
1300 EXAMP0  JSR WAIT           ;A 20662 CYCLE DELAY
1310         BIT SW1            ;IF HIGH BIT SET
1320         BMI EXAMP1         ;  THEN MOVE SPRITE LEFT
1330         LDX #$01           ;  ELSE MOVE IT RIGHT 1 PIXEL
1340         LDA #$00           ;0 INDICATES SPRITE 0
1350         JSR MVSX           ;MOVE THE SPRITE
1360         LDA #$00
1370         JSR SPONX          ;SEE IF ANY OF THE SPR STILL ON SCREEN
1380         BCS EXAMP3         ;  THEN BRANCH TO MOVE UP/DOWN
1390         BCC EXAMP2         ;  ELSE REVERSE X DIRECTION FIRST
1400 ;
1410 EXAMP1  LDX #$FF           ;MOVE SPRITE LEFT 1 PIXEL
1420         LDA #$00           ;0 INDICATES SPRITE 0
1430         JSR MVSX           ; MOVE THE SPRITE
1440         LDA #$00
1450         JSR SPONX          ;IF ANY OF THE SPR STILL ON SCREEN
1460         BCS EXAMP3         ;  THEN BRANCH TO MOVE UP/DOWN
1470 ;
1480 EXAMP2  LDA SW1            ;  ELSE REVERSE X DIRECTION FIRST
1490         EOR #$FF
1500         STA SW1
1510 ;
1520 ;
1530 EXAMP3  BIT SW2            ;IF HIGH BIT SET
1540         BMI EXAMP4         ;  THEN BRANCH TO MOVE SPRITE UP
1550         LDY #$01           ;  ELSE MOVE IT DOWN 1 PIXEL
```

```
1560          LDA #$00        ;0 INDICATES SPRITE 0
1570          JSR MVSY        ;MOVE THE SPRITE
1580          BCS EXAMP0      ;IF SPR ON SCREEN THEN CYCLE AGAIN
1590          BCC EXAMP5      ;  ELSE REVERSE Y DIRECTION FIRST
1600 ;
1610 EXAMP4   LDY #$FF        ;MOVE SPRITE UP 1 PIXEL
1620          LDA #$00        ;0 INDICATES SPRITE 0
1630          JSR MVSY        ;MOVE THE SPRITE
1640          BCS EXAMP0      ;IF SPR ON SCREEN THEN CYCLE AGAIN
1650 ;
1660 EXAMP5   LDA SW2         ;  ELSE REVERSE Y DIRECTION FIRST
1670          EOR #$FF
1680          STA SW2
1690          JMP EXAMP0      ;  AND THEN START CYCLE AGAIN
1700 ;
1710 ;
1720 ;
1730 MVSX     ASL A           ;.Y=2*SPR#
1740          TAY
1750          TXA             ;.A=2'S COMPLEMENT OFFSET
1760          ADC SP0X,Y
1770          STA SP0X,Y
1780          TXA
1790          BMI MVSX2       ;2'S COMPLEMENT OFFSET<0?
1800          BCC MVSX1       ;NO CARRY FROM ADC ABOVE...SO EXIT
1810 MVSX0    TYA             ;CORRECT THE MSB OF SPRITE X-COORD
1820          LSR A           ;.A=INT(.A/2)
1830          TAY             ;.Y=.A
1840          LDA MSIGX       ;MSIGX IS VIC REGISTER OF X MSB'S
1850          EOR PWR2,Y      ;PWR2 IS A TABLE OF THE POWERS OF TWO
1860          STA MSIGX
1870 MVSX1    RTS
1880 ;
1890 MVSX2    BCC MVSX0
1900          RTS
1910 ;
1920 ;
1930 SPONX    TAX             ;AT START .A SHOULD CONTAIN SPR#
1940          ASL A           ;THE CARRY FLAG IS CLEARED
1950          TAY             ;.Y=2*SPR#
1960          LDA MSIGX       ;MSIGX IS VIC REGISTER OF X MSB'S
1970          AND PWR2,X      ;PWR2 IS A TABLE OF POWERS OF 2
1980          BNE SPONX1      ;IF X>$FF THEN CMP LOW ORDER PART
1990          LDA SP0X,Y      ;SEE IF 8 LSB'S OF X ARE ZEROS
2000          BEQ SPONX0      ;IF X=0 THEN EXIT
2010          SEC             ;  ELSE IF 0<X<$FF THEN SET CARRY
2020 SPONX0   RTS             ;EXIT
2030 ;
2040 SPONX1   LDA #$57        ;X>$157? ($157 IS LAST X ON SCREEN)
2050          CMP SP0X,Y      ;THE CARRY FLAG IS UPDATED
2060          RTS
2065 ;
2070 ;
2080 MVSY     ASL A           ;AT START .A SHOULD BE LOADED WITH SPR#
2090          TAX             ;.X=2*SPR#
2100          TYA             ;OFFSET MOVED INTO .A
```

```
2110         ADC SP0Y,X      ;OFFSET IS ADDED TO SPR Y POSITION
2120         STA SP0Y,X      ;SUM IS NEW Y POSITION
2130         CMP #$1D        ;IF Y<$1D THEN C=0
2140         BCC MVSY0       ;  (AND BRANCH TO EXIT)
2150         LDA #$F9        ;  ELSE IS Y>$F9? (LAST Y ON SCREEN)
2160         CMP SP0Y,X      ;CARRY IS UPDATED ACCORDINGLY
2170 MVSY0   RTS             ;EXIT
2180 ;
2190 ;
2200 WAIT    LDX #$FF        ;WAITS 20662 MACHINE CYCLES
2210 WAIT1   LDY #$0F
2220 WAIT2   DEY
2230         BNE WAIT2
2240         DEX
2250         BNE WAIT1
2260         RTS
2270 ;
2280 ;
2290 PWR2    .BYTE $01,$02,$04,$08,$10,$20,$40,$80
2300 ;
2310 .END
```

B. EXAMPLE 2

This example demonstrates the use of UNIMVX and its auxiliary subroutines. When executed, it will display a solid x-expanded sprite on the screen. In the upper-right corner of the screen is shown the x position of the sprite in hexadecimal notation. By pressing the cursor-right key, the user can move the sprite right across (and around) the screen. Because the universal x-movement routine is used, this example checks for the television standard used. Note that on a PAL television the sprite moves from $1F7 to 0.

NOTES:

1) This example steps on zero page and contains the BRK command. For these reasons it should be run from VICMON (with an alternate zero page enabled).

2) The user is advised to experiment with the routine by changing the two's complement offset in line 1590 of the source listing to different values.

```
1000 .PAGE 'EXAMPLE2'
1010 ;
1020 ; EXAMPLE OF UNIVERSAL SPRITE MOVE
1030         *=$0010
1040 ;VARIABLES
1050 OSH     *=*+1   ;OFFSET HIGH
1060 OSL     *=*+1   ;OFFSET LOW
1070 MODH    *=*+1   ;MODULUS HIGH
1080 MODL    *=*+1   ;MODULUS LOW
1090 TX      *=*+1
1100 TA      *=*+1
1110 SMSB    *=*+8   ;AUXILIARY SPRITE MSB BYTES
1120 ;
1130 ;CONSTANTS
1140 SLSB=$D000      ;VIC REGISTER SPRITE LSB BYTES
```

```
1150 MSIGX=$D010     ;VIC REGISTER SPRITE MSB BITS
1160 VIC=$D000
1170 ;
1180          *=$C000
1190 ;
1200 EXAM2    SEI               ;DISABLE IRQS
1210          LDX #$01          ;SET UP PAL MODULUS MSB
1220          LDY #$F8          ;SET UP PAL MODULUS LSB
1230 EX0      LDA VIC+$11       ;LOOK AT RASTER MSB. IS IT A 1 ?
1240          BPL EX0           ;IF 0 THEN RASTER < 256. SO LOOK AGAIN
1250 EX1      LDA #$08          ;RASTER > 255 ...BUT...
1260          CMP VIC+$12       ;IS IT GREATER THAN 264 ?
1270          BCC EX2           ;IF YES THEN BRANCH. TV STD = PAL !!
1280          LDA VIC+$11       ;IF NO THEN CHECK FOR MSB OF RASTER = 1
1290          BMI EX1           ;IF YES THEN GOTO EX1
1300          LDX #$02          ;SET UP NTSC MODULUS MSB. TV STD = NTSC
1310          LDY #$00          ;SET UP NTSC MODULUS LSB
1320 EX2      STX MODH          ;STORE IN MODULUS FOR FUTURE USE
1330          STY MODL
1340          CLI
1350 ;
1360 ;
1370 ;
1380          LDA #1            ;ENABLE AND EXPAND SPRITE 0
1390          STA VIC+21
1400          STA VIC+29
1410          LDA #$7F          ;SET SPRITE Y POSITION TO 127
1420          STA VIC+1
1430          LDA #$80          ;SET SPRITE POINTER TO 128
1440          STA $07F8
1450          LDA #0            ;CLEAR OUT SLSB , SMSB & MSIGX
1460          STA SLSB
1470          STA MSIGX
1480          STA SMSB
1490          LDX #62           ;SET SPRITE TO SOLID BLOCK
1500          LDA #$FF
1510 EX3      STA $2000,X       ;NOTE:- IF SPRITE POINTER = 128
1520          DEX               ; THEN SPRITE ADDRESS = $2000
1530          BPL EX3
1540 ;
1550 ;
1560 ;
1570 EX4      JSR VTAMX         ;TRANSFER MSIGX BITS TO SMSB BYTES
1580          LDA #0            ;LOAD A WITH SPRITE NO.
1590          LDX #$01          ;LOAD X WITH 2'S COMPLEMENT OFFSET (+1)
1600          JSR UNIMVX        ;MOVE SPR MODULO (504-PAL OR 512-NTSC)
1610          JSR ATVMX         ;TRANSFER SMSB BYTES TO MSIGX BITS
1620          LDA SMSB          ;CONVERT SPRITE X COORDINATE TO ASCII
1630          LDY #0            ;HEX CHARACTERS FOR DISPLAY ON TV
1640          JSR BTH           ;BINARY TO HEX (SCREEN ASCII) CONVERTER
1650          LDA VIC
1660          LDY #2
1670          JSR BTH
1680 EX5      JSR $FFE4         ;KERNAL KEY SCANNER
1690          CMP #$1D          ;LOOK FOR 'CUROR-RIGHT' KEY
1700          BEQ EX4           ;IF 'CURSOR-R' KEY THEN MOVE SPRITE
```

```
AGAIN
             CMP #$0D         ;LOOK FOR 'RETURN' KEY
             BNE EX5          ;IF NOT DOWN THEN SCAN KBD AGAIN
             BRK              ;IF DOWN THEN EXIT TO VICMON
             BRK
      ;
      ;         CONVERT BYTE TO TWO SCREEN HEX CHARACTERS
      ;
      BTH    PHA
             AND #$0F
             JSR CONV
             STA $0421,Y
             PLA
             LSR A
             LSR A
             LSR A
             LSR A
             JSR CONV
             STA $0420,Y
             RTS
      ;
      ;         CONVERT NYBBLE TO SCREEN CHARACTER HEX
      ;
      CONV   CMP #10
             BCC CONV1
             SBC #9
             RTS
      CONV1  ORA #$30
             RTS
      ;
      ;
      ;        UNIVERSAL MOVE SPRITE IN X DIRECTIONS
      ;
      ;     A REG=SPRITE NO.    X REG=OFFSET (2'S COMPLEMENT FORM)
      ;
      UNIMVX STX TX           ;PROTECT X
             STX OSL          ;SET UP OFFSET LOW
             TAX              ;A=X
             ASL A            ;A=2*A
             TAY              ;Y=A
             LDA #0           ;CLEAR OFFSET HIGH
             STA OSH
             CLC
             LDA OSL          ;CHECK FOR NEGATIVE OFFSET
             BPL UMX0         ;IF POSITIVE THEN BRANCH
             EOR #$FF         ;PERFORM 2'S COMPLEMENT OPERATION
             ADC #1
             STA OSL          ;PUT RESULTS IN OSL AND THEN
             SEC              ;FORM THE MODULAR COMPLEMENT OF
             LDA MODL         ;THE OFFSET BY SUBTRACTING IT
             SBC OSL          ;FROM THE MODULUS
             STA OSL
             LDA MODH         ;PAL 504 , NTSC 512
             SBC OSH
             STA OSH
             CLC
```

```
2260 UMX0    LDA SLSB,Y      ;ADD OFFSET TO SPRITE X
2270         ADC OSL         ;[SMSB+X,SLSB+Y]=...
2280         STA SLSB,Y      ;  ...[SMSB+X,SLSB+Y]+[OSH,OSL]
2290         LDA SMSB,X
2300         ADC OSH
2310         STA SMSB,X
2320         SEC             ;IS THE SUM >= MODULUS?
2330         LDA SLSB,Y      ;CHECK BY SUBTRACTING
2340         SBC MODL
2350         STA TA          ;CATCH FOR LATER USE
2360         LDA SMSB,X
2370         SBC MODH
2380         BCC UMX1        ;IF SUM < MOD THEN EXIT
2390         STA SMSB,X      ;OTHERWISE CORRECT X-COORD
2400         LDA TA
2410         STA SLSB,Y
2420 UMX1    TYA             ;RESTORE A REG
2430         LSR A
2440         LDX TX          ;RESTORE X REG
2450         RTS
2460 ;
2470 ;          TRANSFER SPRITE MSB BYTES TO MSIGX BITS
2480 ;
2490 ATVMX   LDX #7
2500 ATV0    LDA SMSB,X
2510         LSR A
2520         ROL MSIGX
2530         DEX
2540         BPL ATV0
2550         RTS
2560 ;
2570 ;          TRANSFER SPRITE MSIGX BITS TO MSB BYTES
2580 ;
2590 VTAMX   LDX #7
2600         LDA MSIGX
2610 VTA0    LSR SMSB,X
2620         ASL A
2630         ROL SMSB,X
2640         DEX
2650         BPL VTA0
2660         RTS
2670 ;
2680 .END
```

SOFTWARE APPLICATION NOTE 1002

Author : Joe McEnerney & Bill Hindorff

Subject : Sprite/Character Utility Operators

Television Standard : NTSC or PAL

Abstract : These routines allow various manipulations of RAM which contains a sprite or a character. Prior to being called, source pointers (SRC1 and SRC2) and a destination pointer (DST) must be set up. Each routine has three entry points...the first is for characters, the second for sprites, and the last for a user-defined block of RAM. The last entry point requires the user to initialize the y register to the total number of bytes to operate on minus one. The routines include: transferring a block of memory, logically ANDing two blocks of memory together and storing the result in a third block, logically ORing two blocks of memory, and logically EORing two blocks of memory. A second set of routines logically ANDs a block of memory with a single byte mask, logically ORs a block of memory with a mask, and logically EORs a block of memory with a mask.

Exposition:

b) Program listing as assembled at $C100.

1) Assembly

```
1000 .PAGE (UTOPR9/14)
1010 ;
1020 *=$61
1030 SRC1 *=*+2
1040 SRC2 *=*+2
1050 DST  *=*+2
1060 MASK *=*+1
1070 *=$C100
1080 ;
1090 ;THESE ROUTINES PERFORM VARIOUS UTILITY
1100 ;OPERATIONS.  INITIALIZING A SOURCE POINTER,
1110 ;AND TWO SPRITES/CHARACTERS, OR TWO SPRITES/CHARS,
1120 ;EOR TWO SPRITES/CHARS, AND SPRITE/CHAR WITH A MASK,
1130 ;OR A SPRITE/CHAR WITH A MASK, EOR SPRITE/CHAR
1140 ;WITH A MASK, FILL A SPRITE/CHAR PATTERN,
1150 ;AND TRANSFER A SPRITE/CHAR PATTERN
1160 ;CHARACTER ENTRY POINTS BEGIN WITH A C, SPRITE
1170 ;ENTRY POINTS TYPICALLY BEGIN WITH AN S WITH
1180 ;NO NUMBER SUFFIX.
```

```
1190 ;
1200 ; AUTHOR JOE MCENERNEY & BILL HINDORFF
1210 ;
1220 ;****************************************
1230 ;* SET THE SRC1,X POINTER TO BE THE ACC
1240 ;* TIMES 64.  X SHOULD BE 0 FOR SRC1,
1250 ;* 2 FOR SRC2, AND 4 FOR DST.
1260 ;****************************************
1270 SPPA
1280       LDY #$00
1290       STY SRC1,X
1300       LSR A           ;MULTIPLY ACC BY 64
1310       ROR SRC1,X      ;AND CREATE LOW / HI
1320       LSR A           ;POINTER
1330       ROR SRC1,X
1340       STA SRC1+1,X
1350       RTS
1360 ;
1370 ;****************************************
1380 ;* SET THE SRC1,X POINTER TO BE THE ACC
1390 ;* TIMES EIGHT.  X IS 0 FOR SRC1, 2 FOR
1400 ;* SRC2, AND 4 FOR DST.
1410 ;****************************************
1420 ;
1430 CHPA
1440       LDY #0
1450       STY SRC1+1,X
1460       ASL A
1470       ROL SRC1+1,X
1480       ASL A
1490       ROL SRC1+1,X
1500       ASL A
1510       ROL SRC1+1,X
1520       STA SRC1,X
1530       RTS
1540 ;
1550 ;****************************************
1560 ;* DUPLICATE SRC2 POINTER AS THE
1570 ;* DESTINATION POINTER DST.
1580 ;****************************************
1590 ;
1600 S2DST
1610       LDA SRC2
1620       STA DST
1630       LDA SRC2
1640       STA DST+1
1650       RTS
1660 ;
1670 ;****************************************
1680 ;* FILL THE PATTERN POINTED TO BY
1690 ;* DST WITH THE VALUE IN THE ACC.
1700 ;* ACC IS TYPICALLY 00 OR FF.
1710 ;****************************************
1720 ;
1730 SETCH
```

```
1740        LDY #7           ;ENTRY FOR CHAR. SET
1750        BNE SETSP1       ;ALWAYS
1760 SETSP                   ;ENTRY FOR SPRITE SET
1770        LDY #62
1780 SETSP1                  ;ENTRY FOR PRE-DEFINED
1790        STA (DST),Y      ;# OF BYTES TO FILL
1800        DEY
1810        BPL SETSP1
1820        RTS
1830 ;
1840 ;*******************************************
1850 ;* TRANSFER THE PATTERN POINTED TO BY SRC1
1860 ;* TO THE LOCATION STARTING AT DST
1870 ;*******************************************
1880 ;
1890 CXFER
1900        LDY #7
1910        BNE SXFER1
1920 SXFER
1930        LDY #62
1940 SXFER1
1950        LDA (SRC1),Y
1960        STA (DST),Y
1970        DEY
1980        BPL SXFER1
1990        ROL A
2000        RTS
2010 ;
2020 ;*******************************************
2030 ;* LOGICAL AND THE PATTERN POINTED TO BY
2040 ;* SRC1 WITH A MASK
2050 ;*******************************************
2060 ;
2070 CANDM
2080        LDY #7
2090        BNE SANDM1
2100 SANDM
2110        LDY #62
2120 SANDM1
2130        LDA (SRC1),Y
2140        AND MASK
2150        STA (DST),Y
2160        DEY
2170        BPL SANDM1
2180        RTS
2190 ;
2200 ;*******************************************
2210 ;* LOGICAL OR THE PATTERN POINTED TO BY
2220 ;* SRC1 WITH A MASK
2230 ;*******************************************
2240 ;
2250 CORM
2260        LDY #7
2270        BNE SORM1
2280 SORM
```

```
2290        LDY #62
2300 SORM1
2310        LDA (SRC1),Y
2320        ORA MASK
2330        STA (DST),Y
2340        DEY
2350        BPL SORM1
2360        RTS
2370 ;
2380 ;******************************************
2390 ;* LOGICAL EOR THE PATTERN POINTED TO BY
2400 ;* SRC1 WITH A MASK
2410 ;******************************************
2420 ;
2430 CEORM
2440        LDY #7
2450        BNE SEORM1
2460 SEORM
2470        LDY #62
2480 SEORM1
2490        LDA (SRC1),Y
2500        EOR MASK
2510        STA (DST),Y
2520        DEY
2530        BPL SEORM1
2540        RTS
2550 ;
2560 ;******************************************
2570 ;* LOGICAL AND THE PATTERN POINTED TO BY
2580 ;* SRC1 WITH THE PATTERN POINTED TO BY
2590 ;* SRC2 AND STORE IT IN DESTINATION DST
2600 ;******************************************
2610 ;
2620 CANDC
2630        LDY #7
2640        BNE SANS0
2650 SANDS
2660        LDY #62
2670 SANS0
2680        CLC
2690 SANS1
2700        LDA (SRC1),Y
2710        AND (SRC2),Y
2720        STA (DST),Y
2730        BEQ SANS2
2740        SEC
2750 SANS2
2760        DEY
2770        BPL SANS1
2780        RTS
2790 ;
2800 ;******************************************
2810 ;* LOGICAL OR THE PATTERN POINTED TO BY
2820 ;* SRC1 WITH THE PATTERN POINTED TO BY
2830 ;* SRC2 AND STORE IT IN DESTINATION DST
```

```
2840 ;*********************************************
2850 ;
2860 CORC
2870        LDY #7
2880        BNE SORS1
2890 SORS
2900        LDY #62
2910 SORS1
2920        LDA (SRC1),Y
2930        ORA (SRC2),Y
2940        STA (DST),Y
2950        DEY
2960        BPL SORS1
2970        RTS
2980 ;
2990 ;*********************************************
3000 ;* LOGICAL EOR THE PATTERN POINTED TO BY
3010 ;* SRC1 WITH THE PATTERN POINTED TO BY
3020 ;* SRC2 AND STORE IT IN DESTINATION DST
3030 ;*********************************************
3040 ;
3050 CEORC
3060        LDY #7
3070        BNE SEORS1
3080 SEORS
3090        LDY #62
3100 SEORS1
3110        LDA (SRC1),Y
3120        EOR (SRC2),Y
3130        STA (DST),Y
3140        DEY
3150        BPL SEORS1
3160        RTS
3170 ;
3180 ;
3190 .END
```

2) Hex dump:

```
.: 2100 A0 00 94 61 4A 76 61 4A 76 61 95 62 60 A0 00 94
.: 2110 62 0A 36 62 0A 36 62 0A 36 62 95 61 60 A5 63 85
.: 2120 65 A5 63 85 66 60 A0 07 D0 02 A0 3E 91 65 88 10
.: 2130 FB 60 A0 07 D0 02 A0 3E B1 61 91 65 88 10 F9 2A
.: 2140 60 A0 07 D0 02 A0 3E B1 61 25 67 91 65 88 10 F7
.: 2150 60 A0 07 D0 02 A0 3E B1 61 05 67 91 65 88 10 F7
.: 2160 60 A0 07 D0 02 A0 3E B1 61 45 67 91 65 88 10 F7
.: 2170 60 A0 07 D0 02 A0 3E 18 B1 61 31 63 91 65 F0 01
.: 2180 38 88 10 F4 60 A0 07 D0 02 A0 3E B1 61 11 63 91
.: 2190 65 88 10 F7 60 A0 07 D0 02 A0 3E B1 61 51 63 91
```

3) Data statements:

```
DATA 160,0,148,97,74,118,97,74,118,97,149
DATA 98,96,160,0,149,98,10,54,98,10,54,98
DATA 10,54,98,149,97,96,165,99,133,101,165
DATA 99,133,102,96,160,7,208,2,160,62,145
DATA 101,136,16,251,96,160,7,208,2,160,62
DATA 177,97,145,101,136,16,249,42,96,160,7
DATA 208,2,160,62,177,97,37,103,145,101,136
DATA 16,247,96,160,7,208,2,160,62,177,97,5
DATA 103,145,101,136,16,247,96,160,7,208,2
DATA 160,62,177,97,69,103,145,101,136,16,247
DATA 96,160,7,208,2,160,62,24,177,97,49,99
DATA 145,101,240,1,56,136,16,244,96,160,7
DATA 208,2,160,62,177,97,17,99,145,101,136
DATA 16,247,96,160,7,208,2,160,62,177,97,81
DATA 99,145,73
```

b) Memory/Register requirements: These routines require 160 bytes of memory and 7 bytes of zero page RAM. They use the accumulator and the x and y registers.

c) Worst case execution time is 1458 cycles (or 1428.84 usecs on a 1.02 MHz system).

d) Prior to using this subroutine the appropriate pointers must be initialized for the subroutine being used. Refer to the subroutine's comments to find out what variables must be pre-set.

e) Example: The following program initializes the three sprite pointers, ORs two sprite patterns together, and places the result in memory starting at the destination pointer.

```
1000 ORTWO
1010       LDA #07F8
1020       LDX #00
1030       JSR SPPA          ;SET SOURCE 1 ADDRESS
1040       LDA #07F9
1050       LDX #02
1060       JSR SPPA          ;SET SOURCE 2 ADDRESS
1070       LDA #07FA
1080       LDX #04
1090       JSR SPPA          ;SET DESTINATION ADDRESS
1100       JSR SORS          ;OR THE TWO PATTERNS AND STORE
1200       RTS
```

SOFTWARE APPLICATION NOTE 1003

Author: Bill Hindorff

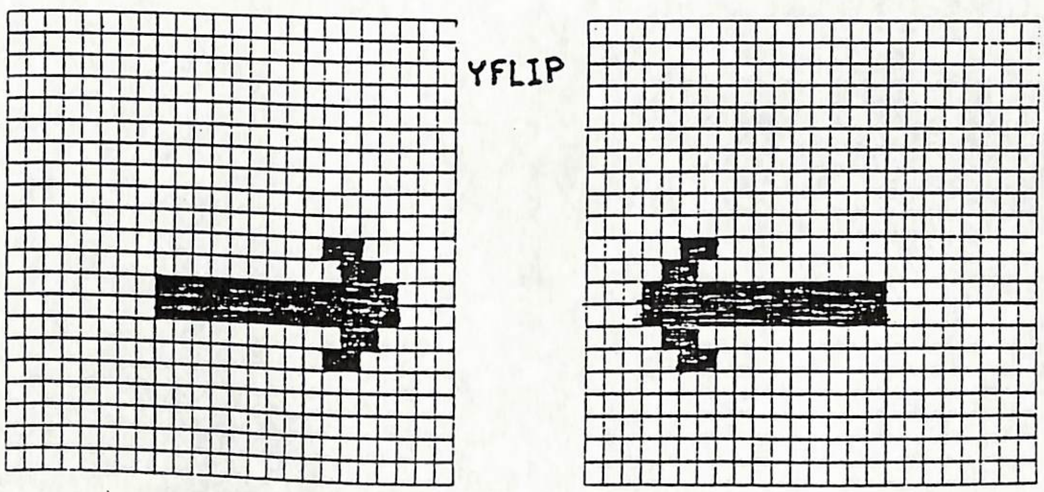
Subject: Sprite pattern mirroring and shifting.

Television Standard: NTSC or PAL

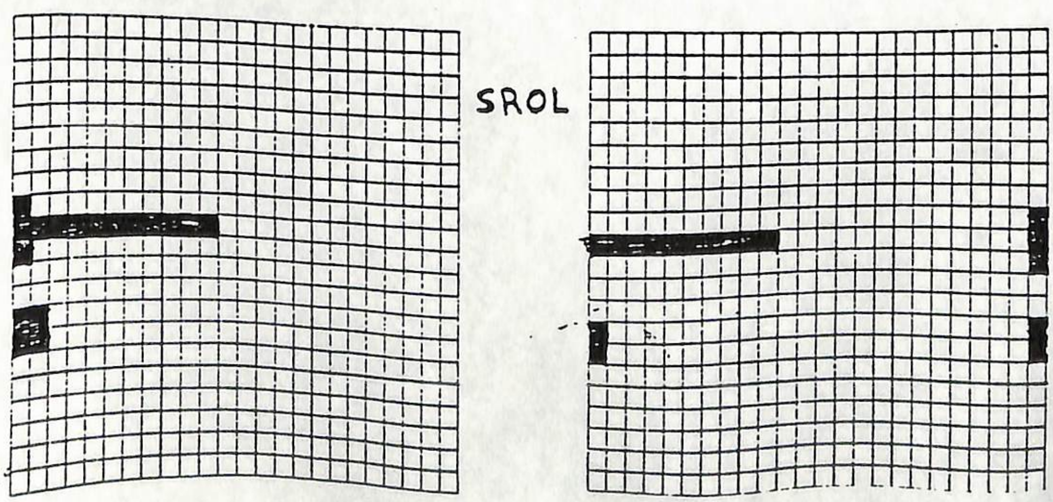
Abstract: These routines take a user defined sprite in RAM and reflect it about its central y-axis, reflect it about its central x-axis, shift/rotate left, and shift/rotate up. Prior to being called, a pointer to the starting address of the sprite data (spr) must be initialized. In the case of shifting or rotating, the x register must also be set with the number of bits to shift.

Exposition:

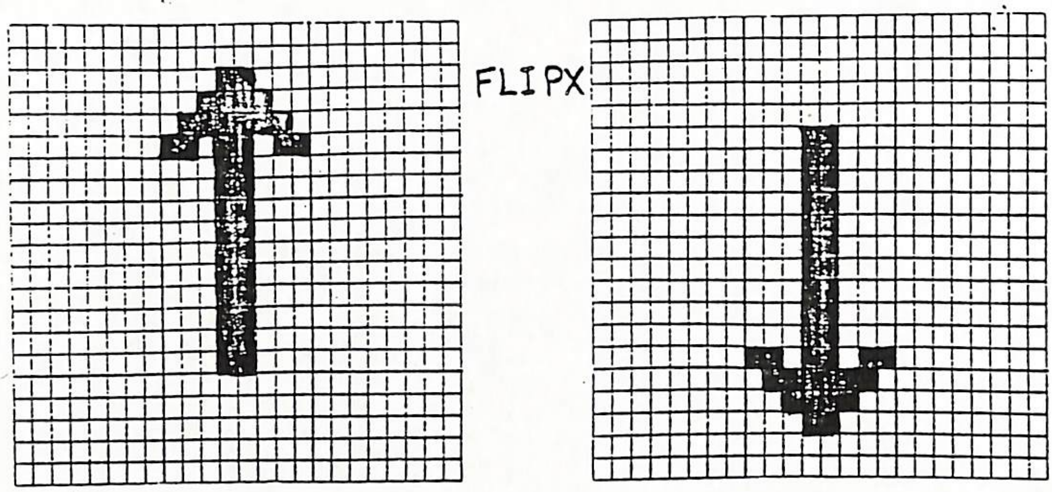
a) Diagrams: The two patterns below illustrate a sprite before being flipped about the y-axis and after being flipped.



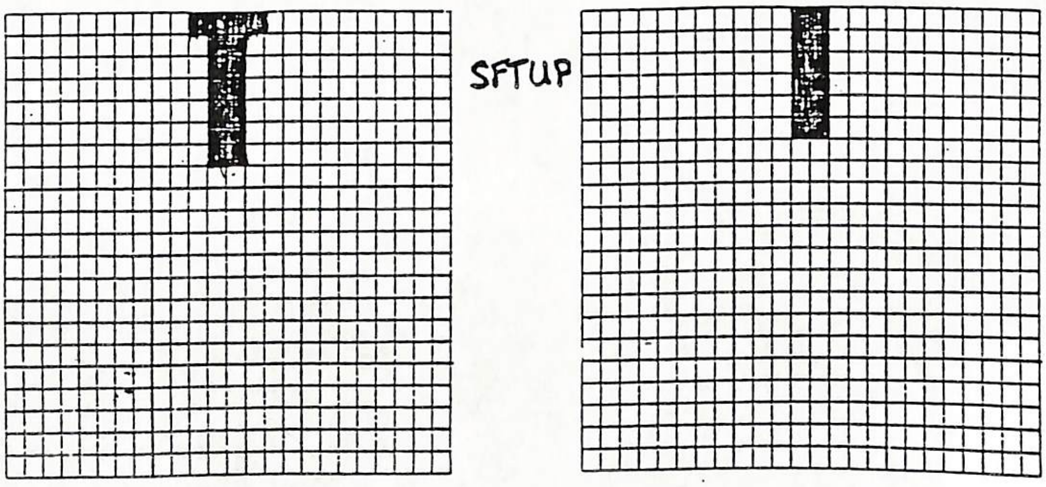
These views show a sprite being rotated left 1 bit.



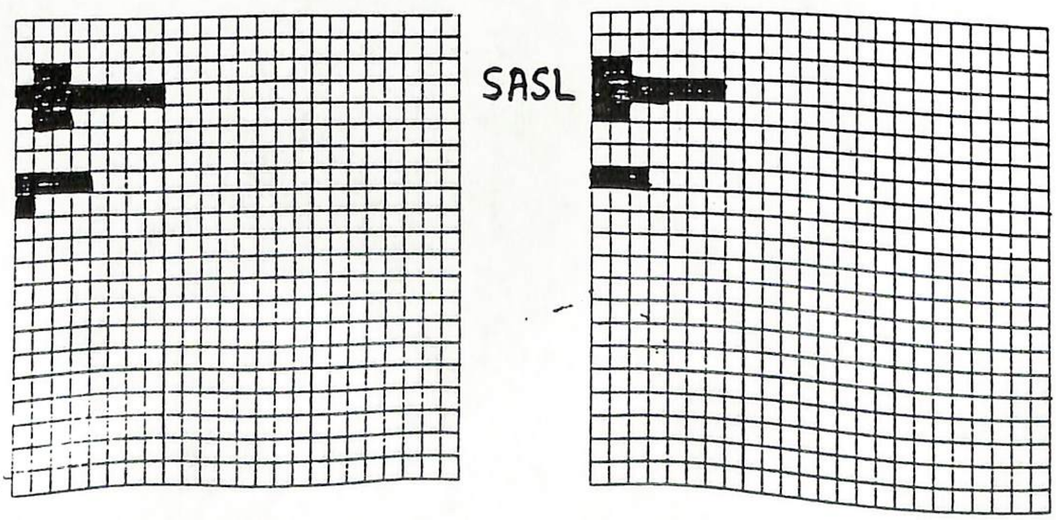
This illustration shows a before and after shot of a sprite being mirrored about the x-axis.



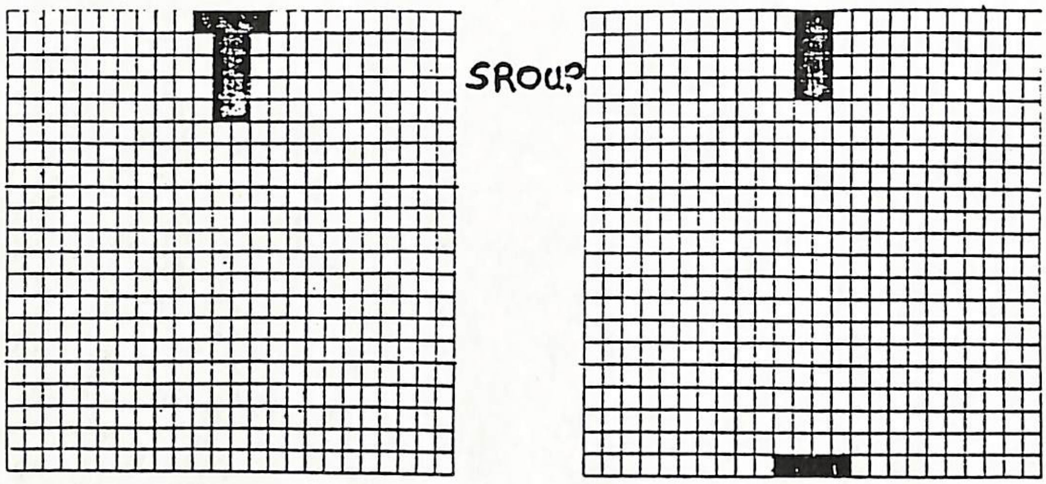
The patterns below represent a sprite which has been shifted up 1 bit (x register = $01).



A before and after view of a sprite being shifted left 1 bit (x register = $01).



A sprite before being rotated up 1 bit, and after being rotated.



b) Program listing as assembled at $C100.

1) Assembly:

```
1000 .PAGE (SPYTRN9/19)
1010 ;
1020 *=$61
1030 SPR *=*+2
1040 SPRH *=*+2
1050 BUFFER *=*+1
1060 *=$C100
1070 ;
1080 ;THIS ROUTINE WILL FLIP THE SPRITE AT ADDRESS SPR
1090 ;ABOUT THE Y-AXIS (RUNNING THRU THE CENTER ,
1100 ;BETWEEN BITS 11 AND 12). SPRH IS EQUAL TO SPR+2.
1110 ;PRIOR TO CALLING THE ROUTINE, THE USER SHOULD
1120 ;INITIALIZE SPR AND SPR+1 TO THE SPRITE'S
1130 ;STARTING ADDRESS IN MEMORY.
1140 ;THE SPRITE SHOULD BE IN HIRES MODE.
1150 ;
1160 ; AUTHOR BILL HINDORFF
1170 ;
1180 YFLIP
1190        LDY #$3C        ;INITIALIZE Y
1200 YFLIPP
1210        LDA SPR         ;GET LOW BYTE OF SPRITE POINTER
1220        CLC
1230        ADC #2          ;CREATE SPRH LOW BYTE
1240        STA SPRH        ;AND SAVE IT
1250        LDX SPR+1       ;GET HI BYTE OF SPRITE POINTER
1260        BCC YFLIP0      ;SHOULD IT BE INCREMENTED
1270        INX             ;YES
1280 YFLIP0
1290        STX SPRH+1      ;SAVE SPRH HIGH BYTE
1300 YFLIP1
```

```
1310        LDA (SPR),Y   ;GET THE LEFT BYTE
1320        STA BUFFER    ;SAVE IT AWAY
1330        LDA (SPRH),Y  ;GET THE RIGHT BYTE
1340        LDX #7
1350 YFLIP2
1360        ROL A         ;FLIP LEFT BYTE
1370        ROR BUFFER    ;AND RIGHT BYTE
1380        DEX
1390        BPL YFLIP2
1400        ROL A
1410        STA (SPRH),Y  ;PUT FLIPPED RIGHT BYTE IN LEFT
1420        LDA BUFFER
1430        STA (SPR),Y   ;PUT FLIPPED LEFT BYTE IN RIGHT
1440        DEY
1450        LDA (SPRH),Y  ;GET CENTER BYTE
1460        STA BUFFER
1470        LDX #7
1480 YFLIP3
1490        ROL A
1500        ROR BUFFER    ;FLIP CENTER BYTE
1510        DEX
1520        BPL YFLIP3
1530        ROL A
1540        STA (SPRH),Y  ;PUT BACK IN CENTER
1550        DEY
1560        DEY           ;POINT TO NEXT ROW
1570        BPL YFLIP1    ;BRANCH IF NOT DONE
1580        RTS
1590 ;
1600 ;*************************************************
1610 ;* THIS SUBROUTINE WILL TAKE A SPRITE PATTERN
1620 ;* WHICH IS LESS THAN 24 BITS WIDE AND ROTATE
1630 ;* IT TO THE LEFT.  THE X REGISTER SHOULD BE
1640 ;* PRE-SET WITH THE NUMBER OF BITS TO ROTATE
1650 ;* AND SPR SHOULD CONTAIN THE LOW/HI POINTER TO
1660 ;* THE STARTING MEMORY LOCATION OF THE SPRITE.
1670 ;*************************************************
1680 ;
1690 SROL
1700        LDY #62
1710 SROL0
1720        STX BUFFER+2  ;ENTRY IF SPRITE IS NOT 21 BITS
1730 SROL1                ;HIGH
1740        LDA (SPR),Y   ;GET THE RIGHT BYTE
1750        TAX           ;SAVE IT
1760        DEY
1770        LDA (SPR),Y   ;GET THE MIDDLE BYTE
1780        STA BUFFER+1  ;SAVE IT
1790        DEY
1800        LDA (SPR),Y   ;GET THE LEFT BYTE
1810        STA BUFFER    ;AND SAVE
1820        TXA           ;START WITH THE RIGHT
1830        LDX BUFFER+2  ;GET NUMBER OF BITS TO SHIFT
1840 SROL2
1850        ASL A         ;SHIFT RIGHT BYTE
1860        ROL BUFFER+1  ;ROTATE MIDDLE BYTE
1870        ROL BUFFER    ;ROTATE LEFT BYTE
```

```
1880        ADC #$00      ;SET LOW OF ACC
1890        DEX
1900        BNE SROL2
1910        INY
1920        INY
1930        STA (SPR),Y   ;STORE RIGHT
1940        DEY
1950        LDA BUFFER+1
1960        STA (SPR),Y   ;AND MIDDLE
1970        DEY
1980        LDA BUFFER
1990        STA (SPR),Y   ;AND LEFT
2000        DEY
2010        BPL SROL1
2020        RTS
2030 ;
2040 ;********************************************
2050 ;* THIS ROUTINE FLIPS A SPRITE ABOUT ITS X
2060 ;* AXIS.  THE USER PROVIDES THE STARTING
2070 ;* POINT OF THE SPRITE TO FLIP IN A POINTER
2080 ;* CALLED SPR
2090 ;********************************************
2100 ;
2110 FLIPX
2120        LDA SPR+1
2130        STA BUFFER+1
2140        STA SPRH+1
2150        LDA SPR
2160        CLC
2170        ADC #$1B         ;CREATE A POINTER TO JUST ABOVE
2180        STA BUFFER       ;CENTER OF SPRITE
2190        BCC FLIPX1
2200        INC BUFFER+1
2210 FLIPX1
2220        CLC
2230        ADC #$6          ;AND A POINTER TO JUST PAST CENTER
2240        STA SPRH
2250        BCC FLIPX2
2260        INC SPRH+1
2270 FLIPX2
2280        LDX #$9          ;COUNTER FOR HALF A SPRITE
2290        STX BUFFER+2
2300 FLIPX3
2310        LDY #$2
2320 FLIPX4
2330        LDA (BUFFER),Y ;GET BYTE OF UPPER SPRITE
2340        TAX              ;SAVE IT
2350        LDA (SPRH),Y     ;GET BYTE OF LOWER SPRITE
2360        STA (BUFFER),Y ;PUT IT UP TOP
2370        TXA
2380        STA (SPRH),Y     ;PUT UPPER DATA DOWN BELOW
2390        DEY              ;DO NEXT IN A LINE
2400        BPL FLIPX4
2410        LDA BUFFER
2420        SEC
2430        SBC #$03         ;MOVE BACK A LINE
2440        STA BUFFER
```

```
2450        BCS FLIPX5
2460        DEC BUFFER+1
2470 FLIPX5
2480        LDA SPRH
2490        CLC
2500        ADC #$03          ;MOVE FORWARD A LINE
2510        STA SPRH
2520        BCC FLIPX6
2530        INC SPRH+1
2540 FLIPX6
2550        DEC BUFFER+2
2560        BPL FLIPX3
2570        RTS
2580 ;
2590 ;*******************************************
2600 ;* THIS ROUTINE SHIFTS A SPRITE PATTERN UP
2610 ;* AFTER IT HAS BEEN FLIPPED ABOUT X
2620 ;* THE USER NEEDS TO SET THE STARTING POINTER
2630 ;* OF THE SPRITE IN ACC AND THE NUMBER OF
2640 ;* LINES TO SHIFT TIMES 3 IN THE X REGISTER
2650 ;*******************************************
2660 ;
2670 SFTUP
2680        LDY #$00
2690        LDA SPR+1
2700        STA SPRH+1
2710        TXA
2720        CLC
2730        ADC SPR           ;CREATE A NEW POINTER TO
2740        STA SPRH          ;THE STARTING LINE TO BE
2750        BCC SFTUP1        ;SHIFTED UP
2760        INC SPRH+1
2770 SFTUP1
2780        LDA (SPRH),Y      ;GET A BYTE TO SHIFT
2790        STA (SPR),Y       ;PUT IT CLOSER TO TOP
2800        INY
2810        INX
2820        CPX #63
2830        BNE SFTUP1
2840        LDA #$00          ;ZERO THE BOTTOM LINE
2850        STA (SPR),Y
2860        DEY
2870        STA (SPR),Y
2880        DEY
2890        STA (SPR),Y
2900        RTS
2910 ;
2920 ;*************************************************
2930 ;* THIS SUBROUTINE WILL TAKE A SPRITE PATTERN
2940 ;* WHICH IS LESS THAN 24 BITS WIDE AND SHIFT
2950 ;* IT TO THE LEFT.  THE X REGISTER SHOULD BE
2960 ;* PRE-SET WITH THE NUMBER OF BITS TO SHIFT
2970 ;* AND SPR SHOULD CONTAIN THE LOW/HI POINTER TO
2980 ;* THE STARTING MEMORY LOCATION OF THE SPRITE.
2990 ;*************************************************
3000 ;
3010 SASL
```

```
3020        LDY #62
3030 SASL0
3040        STX BUFFER+2     ;ENTRY IF SPRITE IS NOT 21.BITS
3050 SASL1                   ;HIGH
3060        LDA (SPR),Y      ;GET THE RIGHT BYTE
3070        TAX              ;SAVE IT
3080        DEY
3090        LDA (SPR),Y      ;GET THE MIDDLE BYTE
3100        STA BUFFER+1     ;SAVE IT
3110        DEY
3120        LDA (SPR),Y      ;GET THE LEFT BYTE
3130        STA BUFFER       ;AND SAVE
3140        TXA              ;START WITH THE RIGHT
3150        LDX BUFFER+2     ;GET NUMBER OF BITS TO SHIFT
3160 SASL2
3170        ASL A            ;SHIFT RIGHT BYTE
3180        ROL BUFFER+1     ;ROTATE MIDDLE BYTE
3190        ROL BUFFER       ;ROTATE LEFT BYTE
3200        DEX
3210        BNE SASL2
3220        INY
3230        INY
3240        STA (SPR),Y      ;STORE RIGHT
3250        DEY
3260        LDA BUFFER+1
3270        STA (SPR),Y      ;AND MIDDLE
3280        DEY
3290        LDA BUFFER
3300        STA (SPR),Y      ;AND LEFT
3310        DEY
3320        BPL SASL1
3330        RTS
3340 ;
3350 ;********************************************
3360 ;* THIS ROUTINE ROTATES A SPRITE PATTERN UP
3370 ;* AFTER IT HAS BEEN FLIPPED ABOUT X
3380 ;* THE USER NEEDS TO SET THE STARTING POINTER
3390 ;* OF THE SPRITE AND THE NUMBER OF LINES
3400 ;* TO ROTATE IN THE X REGISTER.
3410 ;********************************************
3420 ;
3430 SROUP
3440        LDA SPR+1
3450        STA SPRH+1
3460        LDA SPR
3470        CLC
3480        ADC #$03
3490        STA SPRH
3500        BCC SROUP1
3510        INC SPRH+1
3520 SROUP1
3530        LDY #$00
3540        LDA (SPR),Y      ;SAVE THE TOP SPRITE LINE
3550        STA BUFFER
3560        INY
3570        LDA (SPR),Y
3580        STA BUFFER+1
```

```
3590         INY
3600         LDA (SPR),Y
3610         STA BUFFER+2
3620         LDY #$00
3630 GROUP2  LDA (SPRH),Y    ;GET NEXT SPRITE LINE DATA
3650         STA (SPR),Y     ;MOVE IT TO CURRENT LINE
3660         INY
3670         CPY #$3C
3680         BNE GROUP2
3690         LDA BUFFER      ;PUT THE TOP LINE ON BOTTOM
3700         STA (SPR),Y
3710         INY
3720         LDA BUFFER+1
3730         STA (SPR),Y
3740         INY
3750         LDA BUFFER+2
3760         STA (SPR),Y
3770         DEX
3780         BNE GROUP1
3790         RTS
3800 .END
```

2) Hex Dump:

```
.: 2100 A0 3C A5 61 18 69 02 85 63 A6 62 90 01 E8 86 64
.: 2110 B1 61 85 65 B1 63 A2 07 2A 66 65 CA 10 FA 2A 91
.: 2120 63 A5 65 91 61 88 B1 63 85 65 A2 07 2A 66 65 CA
.: 2130 10 FA 2A 91 63 88 88 10 D7 60 A0 3E 86 67 B1 61
.: 2140 AA 88 B1 61 85 66 88 B1 61 85 65 8A A6 67 0A 26
.: 2150 66 26 65 69 00 CA D0 F6 C8 C8 91 61 88 A5 66 91
.: 2160 61 88 A5 65 91 61 88 10 D5 60 A5 62 85 66 85 64
.: 2170 A5 61 18 69 1B 85 65 90 02 E6 66 18 69 06 85 63
.: 2180 90 02 E6 64 A2 09 86 67 A0 02 B1 65 AA B1 63 91
.: 2190 65 8A 91 63 88 10 F3 A5 65 38 E9 03 85 65 B0 02
.: 21A0 C6 66 A5 63 18 69 03 85 63 90 02 E6 64 C6 67 10
.: 21B0 D7 60 A0 00 A5 62 85 64 8A 18 65 61 85 63 90 02
.: 21C0 E6 64 B1 63 91 61 C8 E8 E0 3F D0 F5 A9 00 91 61
.: 21D0 88 91 61 88 91 61 60 A0 3E 86 67 B1 61 AA 88 B1
.: 21E0 61 85 66 88 B1 61 85 65 8A A6 67 0A 26 66 26 65
.: 21F0 CA D0 F8 C8 C8 91 61 88 A5 66 91 61 88 A5 65 91
.: 2200 61 88 10 D7 60 A5 62 85 64 A5 61 18 69 03 85 63
.: 2210 90 02 E6 64 A0 00 B1 61 85 65 C8 B1 61 85 66 C8
.: 2220 B1 61 85 67 A0 00 B1 63 91 61 C8 C0 3C D0 F7 A5
.: 2230 65 91 61 C8 A5 66 91 61 C8 A5 67 91 61 CA D0 D4
.: 2240 60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

b) Data Statements:

```
DATA 160,60,165,97,24,105,2,133,99,166,98,144,1,232,134
DATA 100,177,97,133,101,177,99,162,7,42,102,101,202,16
DATA 250,42,145,99,165,101,145,97,136,177,99,133,101,162
DATA 7,42,102,101,202,16,250,42,145,99,136,136,16,215,96
DATA 160,62,134,103,177,97,170,136,177,97,133,102,136,177
DATA 97,133,101,138,166,103,10,38,102,38,101,105,0,202
DATA 208,246,200,200,145,97,136,165,102,145,97,136,165
DATA 101,145,97,136,16,213,96,165,98,133,102,133,100,165
DATA 97,24,105,27,133,101,144,2,230,102,24,105,6,133,99
DATA 144,2,230,100,162,9,134,103,160,2,177,101,170,177,99
DATA 145,101,138,145,99,136,16,243,165,101,56,233,3,133
DATA 101,176,2,198,102,165,99,24,105,3,133,99,144,2,230
DATA 100,198,103,16,215,96,160,0,165,98,133,100,138,24,101
DATA 97,133,99,144,2,230,100,177,99,145,97,200,232,224,63
DATA 208,246,169,0,145,97,136,145,97,136,145,97,96,160,62
DATA 134,103,177,97,170,136,177,97,133,102,136,177,97,133
DATA 101,138,166,103,10,38,102,38,101,202,208,248,200,200
DATA 145,97,136,165,102,145,97,136,165,101,145,97,136,16
DATA 215,96,165,98,133,100,165,97,24,105,3,133,99,144,2,230
DATA 100,160,0,177,97,133,101,200,177,97,133,102,200,177,97
DATA 133,103,160,0,177,99,145,97,200,192,60,208,247,165,101
DATA 145,97,200,165,102,145,97,200,165,103,145,97,202,208
DATA 212,96
```

c) Memory/Register requirements: This routine requires 321 ($141) bytes of memory. It uses the accumulator, X, and Y registers and 7 bytes of storage.

d) Worst case execution time is 1082*Z+29 cycles for the SROUP routine. Where Z is equal to the number of times the sprite is rotated up.

e) Limitations: To mirror a multi-color sprite about the y-axis, the yflip routine is called then multi-color location 37 ($25) and the sprite color location 39 to 46 ($27 to $2e) are interchanged. Note that this will only work for one sprite unless all sprites have the same coloring. Also, the sprite pattern must be in RAM.

f) Prior to using these subroutines a pointer to the beginning of the sprite (called spr) must be set-up (see the SPPA routine of Apnote #1002). The shift routines also require that the x register be conditioned with the number of bits to shift.

g) Example: This example copies a sprite pattern to RAM then flips it about the x-axis. After the pattern is flipped, it is shifted up to its original location within the sprite matrix.

```
FLIP
     LDA #$00
     STA $07F8
     LDX #$00
     JSR SPPA
     LDY #$3F
FLIP1
     LDA SPDATA,Y
     STA ($61),Y
     DEY
     BPL FLIP1
     LDA #$90
     STA $D000
     STA $D001
     LDA #$01
     STA $D015
FLIP2
     JSR FLIPX
     LDX #$0D
     JSR SROUP
     LDX #$00
     LDY #$00
FLIP3
     DEY
     BNE FLIP3
     DEX
     BNE FLIP3
     JMP FLIP2
```

SOFTWARE APPLICATION NOTE 1004

AUTHORS : Andy Finkel and Joe McEnerney

SUBJECT : Sprite Collision Detection

TELEVISION : NTSC or PAL

ABSTRACT

The detection of collisions is often an important part of computer and video games. The 6566/6567 system supports two types of collisions: sprite to sprite collisions and sprite to screen data collisions. In general, a collision is defined as the instance when non-transparent sprite data coincides with either non-transparent data on another sprite or non-transparent data on the screen.

A. SPRITE TO SPRITE COLLISIONS

ABSTRACT

This routine will interpret the sprite collision and sprite position data provided by the VIC-II 6566/6567 chip to determine which sprites are actually in collision. The collision detection register on the 6566/6567 chip is used to determine which sprites COULD be involved in a collision at the time this routine is called. Then the X and Y positions of the sprites and the size of the player sprite are used to decide which sprites are actually involved in the collision.

EXPOSITION

The routine is called with one sprite defined as the PLAYER sprite. All collisions are reported in reference to the player. To increase the accuracy of collision detection, the SIZE of the player (the highest X and Y values that are non-transparent in this sprite) is also passed to the collision detection routine.

The detection routine returns the number of collisions that the player is involved in. If the player in involved in NO collisions a zero will be returned, EVEN IF THERE ARE OTHER COLLISIONS ON THE SCREEN THAT THE PLAYER IS NOT INVOLVED IN. The sprite numbers of the other sprites involved in the collision are returned in the 7 byte RESULTS array. Up to 7 collisions are possible.

This routine should be called at the end of the frame (during vertical blanking). If a raster driven interrupt routine is being used, this routine can easily become part of that routine.

SOURCE LISTING

```
1000 .PAG   'BANG'
1010 ;
1020 ;**************************************
1030 ;*          COLLISION DETECTION
1040 ;*                ROUTINE
1050 ;*
1060 ;* CALL THIS ROUTINE WITH THE .X REGISTER
1070 ;* CONTAINING THE NUMBER OF THE SPRITE
1080 ;* TO CHECK ON
1090 ;*
1100 ;* RETURNS COUNT (NUMBER OF COLLISIONS)
1110 ;*         RESULT (A 7 BYTE ARRAY OF   )
1120 ;*                (SPRITE NUMBERS      )
```

```
;*
;****************************************
;
;
;VARIABLES
;
*=$A3
COUNT  *=*+1                   ;NUMBER OF VALID COLLISIONS
PLAYER *=*+1                   ;SPRITE TO CHECK ON (PLAYER SPRITE)
PXMAX  *=*+1                   ;MAXIMUM SIZE OF PLAYER IN X DIRECTION
PYMAX  *=*+1                   ;MAXIMUM SIZE OF PLAYER IN Y DIRECTION
COLIDE *=*+1                   ;TEMPORARY STORAGE FOR COLLISION REGISTER
TX     *=*+2                   ;TEMPORARY SPRITE X POS STORAGE
TY     *=*+1                   ;TEMPORARY SPRITE Y POS STORAGE
;
PLX    *=*+2                   ;PLAYER SPRITE X POSITION
PLY    *=*+1                   ;PLAYER SPRITE Y POSITION
TEMP   *=*+1
*=$F7
RESULT *=*+7                   ;SPRITE NUMBERS OF VALUE COLLISIONS
;
;CONSTANTS
;
SPRX   =$D000
SPRY   =$D001
MSBX   =$D010
SPRSPR =$D01E
;
*=$3000
;
BANG   LDA #$00                ;SET NUMBER OF COLLISIONS TO 0
       STA    COUNT
;
       LDA    SPRSPR           ;READ SPRITE-SPRITE COLLISION REG
       STA    COLIDE           ;SAVE IT
;
       STX    PLAYER
       LDA    DOTS,X
       AND    COLIDE           ;PLAYER IN A COLLISION
       BEQ    EXIT             ;NO
;
       JSR    ADDR             ;GET PLAYER POSITION
;
       STA    PLX
       LDA    TX+1
       STA    PLX+1
       LDA    TY
       STA    PLY
;
       LDX    #7               ;DO EVERYBODY
AGAIN  CPX PLAYER              ;EXCEPT SELF
       BEQ    P1
;
       LDA    DOTS,X           ;IS THIS ONE IN A COLLISION ?
       AND    COLIDE
       BEQ    P1               ;NO
```

```
;
         JSR    ADDR           ;GET X,Y POSITION OF THE SPRITE
;
         SEC
         SBC    PLX            ;COMPARE LOW ORDER X BYTES
         TAY                   ;SAVE IT FOR A WHILE
;
         LDA    TX+1           ;NOW THE HIGH ORDER
         SBC    PLX+1
         BCS    COMP           ;TX+1>=PLX+1- NO PROBLEMS
;
         STA    TEMP           ;SAVE HIGH ORDER BITS
         TYA                   ;GET LOW ORDER BITS
         EOR    #$FF           ;GET 2'S COMP (CARRY IS CLEAR)
         ADC    #1
         TAY                   ;NEW SAVE LOW ORDER BITS
;
         LDA    TEMP           ;BRING HIGH ORDER BITS BACK
         EOR    #$FF
         ADC    #0
;
COMP     BNE P1                ;NO COLLISION
         TYA                   ;GET LOW ORDER BITS
         CMP    PXMAX          ;IN RANGE ?
         BCS    P1             ;NO COLLISION
;
         SEC                   ;NOW CHECK Y
         LDA    TY
         SBC    PLY
         BCS    COMPY          ;CHECK IF IT IS NEGATIVE
;
         EOR    #$FF           ;IT IS, SO FLIP IT
         ADC    #1             ;(CARRY IS SET)
;
COMPY    CMP PYMAX             ;COLLISION ?
         BCS    P1             ;NO
;
;* WE HAVE A COLLISION *
;
         INC    COUNT          ;INCREASE # OF COLLISIONS
         LDY    COUNT          ;AND SAVE IT IN THE
         STX    RESULT-1,Y     ;RESULTS TABLE
;
P1       DEX
         BPL    AGAIN
;
EXIT     RTS
;
;**************************
;* CALC SPRITE LOCATION
;* SUBROUTINE
;**************************
;
ADDR     TXA                   ;GET OFFSET TO SPRITE POSITIONS
         ASL    A
         TAY
```

```
2250 ;
2260         LDA    SPRY,Y            ;GET Y COORD
2270         STA    TY                ;SAVE IT
2280 ;
2290         LDA    MSBX              ;GET SPRITE HIGH ORDER BIT
2300         AND    DOTS,X            ;IS IT SET ?
2310         BEQ    SKIP              ;NO
2320         LDA    #1
2330 SKIP    STA TX+1                 ;SAVE IT
2340 ;
2350         LDA    SPRX,Y            ;GET X LOW ORDER COORD
2360         STA    TX                ;SAVE IT
2370         RTS
2380 ;
2390 DOTS    .BYT $01,$02,$04,$08,$10,$20,$40,$80
2400 ;
2410 .END
```

HEX DUMP

```
.:  3000   A9 00 85 A3 AD 1E D0 85
.:  3008   A7 86 A4 BD 81 30 25 A7
.:  3010   F0 54 20 67 30 85 AB A5
.:  3018   A9 85 AC A5 AA 85 AD A2
.:  3020   07 E4 A4 F0 3E BD 81 30
.:  3028   25 A7 F0 37 20 67 30 38
.:  3030   E5 AB A8 A5 A9 E5 AC B0
.:  3038   0E 85 AE 98 49 FF 69 01
.:  3040   A8 A5 AE 49 FF 69 00 D0
.:  3048   1A 98 C5 A5 B0 15 38 A5
.:  3050   AA E5 AD B0 04 49 FF 69
.:  3058   01 C5 A6 B0 06 E6 A3 A4
.:  3060   A3 96 F6 CA 10 BB 60 8A
.:  3068   0A A8 B9 01 D0 85 AA AD
.:  3070   10 D0 3D 81 30 F0 02 A9
.:  3078   01 85 A9 B9 00 D0 85 A8
.:  3080   60 01 02 04 08 10 20 40
.:  3088   80
```

DATA STATEMENTS

```
DATA   169,  0,  133,  163,  173,  30,  208,  133
DATA   167,  134,  164,  187,  129,  48,  37,  167
DATA   240,  84,  32,  103,  48,  133,  171,  165
DATA   169,  133,  172,  165,  170,  133,  173,  162
DATA   7,  228,  164,  240,  62,  189,  129,  48
DATA   37,  167,  240,  55,  32,  103,  48,  56
DATA   229,  171,  168,  165,  169,  229,  172,  176
DATA   14,  133,  174,  152,  73,  255,  105,  1
DATA   168,  165,  174,  73,  255,  105,  0,  208
DATA   26,  152,  177,  165,  176,  21,  56,  165
DATA
```

```
DATA   170, 229, 173, 176, 4, 73, 255, 105
DATA   1, 197, 166, 176, 6, 230, 163, 164
DATA   163, 150, 246, 202, 16, 187, 96, 130
DATA   10, 168, 185, 1, 208, 133, 170, 173
DATA   16, 208, 61, 129, 48, 240, 2, 169
DATA   1, 133, 167, 185, 0, 208, 133, 168
DATA   96, 1, 2, 4, 8, 16, 32, 64
DATA   128
```

MEMORY/REGISTER REQUIREMENTS

This routine uses the accumulator, the .X and the .Y registers, and 2 bytes on the stack. 17 locations on zero page are needed. The routine takes 136 bytes of memory.

WORST CASE EXECUTION TIME

1055 cycles (1034 microseconds on a 1.02 Mhz system)

EXAMPLE:

```
E1      LDA #23         ;SET PLAYER SIZE
        STA PXMAX
        LDA #21
        STA PLYMAX
;
        LDX #0          ;CHECK ON SPRITE #0 COLLISIONS
        JSR BANG
;
        LDA COUNT       ;ANY COLLISIONS ?
        BEQ SKIP        ;NO
;
AGAIN   LDX COUNT       ;BLOW UP EVERYONE IN COLLISION
        LDA RESULT,X    ;GET NUMBER OF SPRITE TO EXPLODE
        JSR EXPLOD      ;(DO EXPLOSION ROUTINE)
        DEC COUNT
        BPL AGAIN
;
        LDA #0          ;NOW BLOW PLAYER UP
        JSR EXPLOD
        RTS
```

NOTE: Remember that sprite to sprite collisions can take place even off screen.

ADDITION: Center Collision Detection

ABSTRACT:

The following modification of the collision detection routine gives the ability to use the center of a sprite to check collisions rather than the top left corner. Instead of the valid collision area centering on the upper left corner of the player sprite the valid collision area will be centered on a point within the sprite.

EXPOSITION

The modification to the routine is confined to two areas: that is, the addition of 2 constants describing the center location of a sprite and modification of the sprite location subroutine to use this center.

SOURCE LISTING

SOURCE LISTING

```
1000 .PAG   'CDANC'
1010 ;
1020 ;****************************************
1030 ;*          COLLISION DETECTION
1040 ;*                ROUTINE
1050 ;*
1060 ;* CALL THIS ROUTINE WITH THE .X REGISTER
```

```
1070 ;* CONTAINING THE NUMBER OF THE SPRITE
1080 ;* TO CHECK ON
1090 ;*
1100 ;* RETURNS COUNT (NUMBER OF COLLISIONS)
1110 ;*           RESULT (A 7 BYTE ARRAY OF  )
1120 ;*                  (SPRITE NUMBERS     )
1130 ;*
1140 ;****************************************
1150 ;
1160 ;
1170 ;VARIABLES
1180 ;
1190 *=$A3
1200 COUNT  *=*+1           ;NUMBER OF VALID COLLISIONS
1210 PLAYER *=*+1           ;SPRITE TO CHECK ON (PLAYER SPRITE)
1220 PXMAX  *=*+1           ;MAXIMUM SIZE OF PLAYER IN X DIRECTION
1230 PYMAX  *=*+1           ;MAXIMUM SIZE OF PLAYER IN Y DIRECTION
1240 COLIDE *=*+1           ;TEMPORARY STORAGE FOR COLLISION REGISTER
1250 TX     *=*+2           ;TEMPORARY SPRITE X POS STORAGE
1260 TY     *=*+1           ;TEMPORARY SPRITE Y POS STORAGE
1270 ;
1280 PLX    *=*+2           ;PLAYER SPRITE X POSITION
1290 PLY    *=*+1           ;PLAYER SPRITE Y POSITION
1300 TEMP   *=*+1
1310 *=$F7
1320 RESULT *=*+7           ;SPRITE NUMBERS OF VALUE COLLISIONS
1330 ;
1340 ;CONSTANTS
1350 ;
1360 SPRX   =$D000
1370 SPRY   =$D001
1380 MSBX   =$D010
1390 SPRSPR =$D01E
1400 XCENT  = 12            ;SPRITE CENTER X
1410 YCENT  = 11            ;SPRITE CENTER Y
1420 ;
1430 *=$3000
1440 ;
1450 CBANG  LDA #$00              ;SET NUMBER OF COLLISIONS TO 0
1460        STA    COUNT
1470 ;
1480        LDA    SPRSPR         ;READ SPRITE-SPRITE COLLISION REG
1490        STA    COLIDE         ;SAVE IT
1500 ;
1510        STX    PLAYER
1520        LDA    DOTS,X
1530        AND    COLIDE         ;PLAYER IN A COLLISION
1540        BEQ    EXIT           ;NO
1550 ;
1560        JSR    ADDR           ;GET PLAYER POSITION
1570 ;
1580        STA    PLX
1590        LDA    TX+1
1600        STA    PLX+1
1610        LDA    TY
1620        STA    PLY
```

```
1630 ;
1640        LDX     #7                    ;DO EVERYBODY
1650 AGAIN  CPX PLAYER                    ;EXCEPT SELF
1660        BEQ     P1
1670 ;
1680        LDA     DOTS,X                ;IS THIS ONE IN A COLLISION ?
1690        AND     COLIDE
1700        BEQ     P1                    ;NO
1710 ;
1720        JSR     ADDR                  ;GET X,Y POSITION OF THE SPRITE
1730 ;
1740        SEC
1750        SBC     PLX                   ;COMPARE LOW ORDER X BYTES
1760        TAY                           ;SAVE IT FOR A WHILE
1770 ;
1780        LDA     TX+1                  ;NOW THE HIGH ORDER
1790        SBC     PLX+1
1800        BCS     COMP                  ;TX+1>=PLX+1  NO PROBLEMS
1810 ;
1820        STA     TEMP                  ;SAVE HIGH ORDER BITS
1830        TYA                           ;GET LOW ORDER BITS
1840        EOR     #$FF                  ;GET 2'S COMP (CARRY IS CLEAR)
1850        ADC     #1
1860        TAY                           ;NEW SAVE LOW ORDER BITS
1870 ;
1880        LDA     TEMP                  ;BRING HIGH ORDER BITS BACK
1890        EOR     #$FF
1900        ADC     #0
1910 ;
1920 COMP   BNE P1                        ;NO COLLISION
1930        TYA                           ;GET LOW ORDER BITS
1940        CMP     PXMAX                 ;IN RANGE ?
1950        BCS     P1                    ;NO COLLISION
1960 ;
1970        SEC                           ;NOW CHECK Y
1980        LDA     TY
1990        SBC     PLY
2000        BCS     COMPY                 ;CHECK IF IT IS NEGATIVE
2010 ;
2020        EOR     #$FF                  ;IT IS, SO FLIP IT
2030        ADC     #1                    ;(CARRY IS SET)
2040 ;
2050 COMPY  CMP PYMAX                     ;COLLISION ?
2060        BCS     P1                    ;NO
2070 ;
2080 ;* WE HAVE A COLLISION *
2090 ;
2100        INC     COUNT                 ;INCREASE # OF COLLISIONS
2110        LDY     COUNT                 ;AND SAVE IT IN THE
2120        STX     RESULT-1,Y            ;RESULTS TABLE
2130 ;
2140 P1     DEX
2150        BPL     AGAIN
2160 ;
2170 EXIT   RTS
2180 ;
```

```
2190 ;*************************
2200 ;* CALC SPRITE LOCATION
2210 ;* SUBROUTINE
2220 ;* (FROM CENTER)
2230 ;*************************
2240 ;
2250 ADDR   TXA                      ;GET OFFSET TO SPRITE POSITIONS
2260        ASL    A
2270        TAY
2280 ;
2290        CLC
2300        LDA    SPRY,Y            ;GET Y COORD
2310        ADC    #YCENT            ;CENTER IN Y DIRECTION
2320        STA    TY                ;SAVE IT
2330 ;
2340        LDA    MSBX              ;GET SPRITE HIGH ORDER BIT
2350        AND    DOTS,X            ;IS IT SET ?
2360        BEQ    SKIP              ;NO
2370        LDA    #1
2380 SKIP   STA TX+1                 ;SAVE IT
2390 ;
2400        CLC
2410        LDA    SPRX,Y            ;GET X LOW ORDER COORD
2420        ADC    #XCENT            ;ADD X CENTER
2430        STA    TX                ;SAVE IT
2440        LDA    TX+1
2450        ADC    #0
2460        STA    TX+1
2470 ;
2480        RTS
2490 ;
2500 DOTS   .BYT $01,$02,$04,$08,$10,$20,$40,$80
2510 ;
2520 .END
```

HEX DUMP

```
.:  3000   A9 00 85 A3 AD 1E D0 85
.:  3008   A7 86 A4 8D 0D 30 25 A7
.:  3010   F0 54 20 67 30 85 AB A5
.:  3018   A9 85 AC A5 AA 85 AD A2
.:  3020   07 E4 A4 F0 3E BD 0D 30
.:  3028   25 A7 F0 37 20 67 30 38
.:  3030   E5 AB AD A5 A9 E5 AC B0
.:  3038   0E 85 AE 78 49 FF 69 01
.:  3040   A8 A5 AE 49 FF 69 00 D0
.:  3048   1A 78 C5 A5 B0 15 38 A5
.:  3050   AA E5 AD B0 04 49 FF 69
.:  3058   01 C5 A6 B0 06 E6 A3 A4
.:  3060   A3 96 F6 CA 10 BB 60 0A
.:  3068   0A A8 18 B9 01 D0 69 08
.:  3070   85 AA AD 10 D0 3D 0D 30
.:  3078   F0 02 A9 01 85 A9 18 B9
```

```
.:  3080   00 D0 69 0C 85 A0 A5 A9
.:  3088   69 00 85 A9 60 01 02 04
.:  3090   08 10 20 40 80
```

DATA STATEMENTS

```
DATA   169,  0,  133,  163,  173,  30,  208,  133
DATA   167,  134,  164,  189,  141,  48,  37,  167
DATA   240,  84,  32,  103,  48,  133,  171,  165
DATA   169,  133,  172,  165,  170,  133,  173,  162
DATA   7,  228,  164,  240,  62,  189,  141,  48
DATA   37,  167,  240,  55,  32,  103,  48,  56
DATA   229,  171,  168,  165,  169,  229,  172,  176
DATA   14,  133,  174,  152,  73,  255,  105,  1
DATA   168,  165,  174,  73,  255,  105,  0,  208
DATA   26,  152,  197,  165,  176,  21,  56,  165
DATA   170,  229,  173,  176,  4,  73,  255,  105
DATA   1,  197,  166,  176,  6,  230,  163,  164
DATA   163,  150,  246,  202,  16,  187,  96,  138
DATA   10,  168,  24,  185,  1,  208,  105,  11
DATA   133,  170,  173,  16,  208,  61,  141,  48
DATA   240,  2,  169,  1,  133,  169,  24,  185
DATA   0,  208,  105,  12,  133,  168,  165,  169
DATA   105,  0,  133,  169,  96,  1,  2,  4
DATA   8,  16,  32,  64,  128
```

MEMORY/REGISTER REQUIREMENTS

This routine uses the accumulator, the .X and the .Y registers, and 2 bytes on the stack. The routine requires 17 bytes of zero page RAM storage. The program takes 148 bytes of memory.

WORST CASE EXECUTION TIME

1245 cycles (1220 microseconds on a 1.02 Mhz system)

LIMITS

It is possible for this routine to report false collisions in the case of two or more sprites having some part in the area defined

by the player size. The following picture makes this clear:

```
       BBBBBBBBBBBBBB
       B            B
       B            B
AAAAAAABAAAAA       B
A      BBBBBBBBBBBBBB
A           A
A       PPPPPPPPPPPPP
A       P   A       P
A       P   A       P
AAAAAAAAPAAAA       P
        P           P
        PPPPPPPPPPPPP
```

As you can see, the player is colliding with sprite A only. But sprite A is colliding both with the player and with sprite B. When the collision detection routine is called, it will report that the player is in collision with BOTH A and B, since each is involved in a collision at the time, and both are within the size of the player sprite.

B. SPRITE TO BACKGROUND COLLISION DETECTION

ABSTRACT

This routine will translate the (X,Y) coordinates of the position of a sprite to background collision into a row/column address on the video matrix. If the sprite center is off screen, an error is flagged. The coordinates of the sprite are in the normal range of 0>=X>=511 and 0>=Y>=255. The row/column address is in the range 0>=X>=39 and 0>=Y>=24.

EXPOSITION

Often in a program it is necessary to know not only that a collision between a sprite and background data took place, but where is took place as well. If only the fact that a collision took place

is needed by a program, this routine is unnecessary, as the sprite background collision register may be used alone in that case. However, if the location of the collision on the screen is needed (to see exactly what the sprite hit, for example), the following routine can be used.

This routine is one approach to the problem. There are some rather important limits to this routine, however. See the LIMITS section (below) for more information.

SOURCE LISTING

```
1000 .PAG   'CORD'
1010 ;
1020 ;**************************
1030 ;* BACKGROUND COLLISION
1040 ;*          ROUTINE
1050 ;**************************
1060 ;
1070 ;CONSTANTS
1080 ;
1090 SPRX   =$D000            ;VIC CHIP REGISTERS
1100 SPRY   =$D001
1110 MSIGX  =$D010
1120 SPRBAK =$D01F
1130 ;
1140 XOFF   =0                ;SCREEN CENTERING CONSTANTS
1150 YOFF   =3
1160 ;
1170 ;
1180 *=$0002
1190 ;
1200 XC      *=*+2            ;2 BYTE X COORDINATE OF SPRITE
1210 YC      *=*+1            ;Y COORDINATE OF SPRITE
1220 ;
1230 ;
1240 *=$3000
1250 ;
1260 ;*****************************************
1270 ;* SPRITE TO BACKGROUND COLLISION ROUTINE
1280 ;*
1290 ;* ENTER THIS ROUTINE WITH THE SPRITE
1300 ;* NUMBER TO CHECK ON IN .Y
1310 ;*
1320 ;* THE COORDINATES ARE RETURNED IN .X (ROW) AND .Y (COL)
1330 ;* CARRY WILL BE CLEAR ON A GOOD RETURN
1340 ;* CARRY WILL BE SET IF SPRITE IS OFF SCREEN
```

```
;* OR IF THERE IS NO COLLISION
;*******************************************
;
CORD    LDA SPRBAK
        AND     DOTS,Y
        BEQ     EXIT            ;NO COLLISION WITH THIS SPRITE
;
        TYA
        ASL     A               ;SET POINTER FOR SPRITE REGISTERS
        TAX
;
        SEC
        LDA     SPRX,X          ;DO X FIRST
        SBC     #24+XOFF        ;SUBTRACT SCREEN OFFSET
        STA     XC
        LDA     MSIGX           ;HANDLE MOB OF SPRITE
        AND     DOTS,Y
        BEQ     SKIP1
;
        LDA     #1
;
SKIP1   SBC #0
        STA     XC+1            ;HANDLE HIGH BYTE
        BCC     EXIT            ;SPRITE IS OFF SCREEN
;
        BEQ     SKIP2           ;SPRITE IS ON LEFT SIDE OF SCREEN
;
        LDA     XC              ;OFF SCREEN ?
        CMP     #$56
        BCS     EXIT            ;YES
;
SKIP2   LSR XC+1                ;DIVIDE BY 8
        LDA     XC
        ROR     A
        LSR     A
        LSR     A
;
        TAY                     ;PREPARE IT FOR RETURN
;
        SEC                     ;NOW DO Y COORD CONVERSION
        LDA     SPRY,X
        SBC     #50+YOFF        ;(WHERE YOFF IS FROM 0-7)
        BCC     EXIT            ;SPRITE IS OFF SCREEN
;
        CMP     #250-50         ;OFF THE BOTTOM ?
        BCS     EXIT            ;YES
;
        LSR     A               ;DIVIDE BY 8
        LSR     A
        LSR     A
;
        TAX                     ;PASS IT BACK IN X
;
        CLC                     ;SHOW GOOD EXIT
        RTS
;
```

```
1910 EXIT    SEC               ;ERROR EXIT
1920         RTS
1930 ;
1940 DOTS    .BYT $01,$02,$04,$08,$10,$20,$40,$80
1950 ;
1960 .END
```

HEX DUMP

```
.:  3000   B9 1F D0 39 47 30 F0 3D
.:  3008   98 0A AA 38 BD 00 D0 E9
.:  3010   18 85 02 AD 10 D0 39 47
.:  3018   30 F0 02 A9 01 E9 00 85
.:  3020   03 90 22 F0 06 A5 02 C9
.:  3028   56 B0 1A 46 03 A5 02 6A
.:  3030   4A 4A A0 38 BD 01 D0 E9
.:  3038   35 90 0A C9 C8 B0 06 4A
.:  3040   4A 4A AA 18 60 38 60 01
.:  3048   02 04 08 10 20 40 80
```

DATA STATEMENTS

```
DATA   185,  31,  208,  57,  71,  48,  240,  61
DATA   152,  10,  170,  56,  189,  0,  208,  233
DATA   24,  133,  2,  173,  16,  208,  57,  71
DATA   48,  240,  2,  169,  1,  233,  0,  133
DATA   3,  144,  34,  240,  6,  165,  2,  201
DATA   86,  176,  26,  70,  3,  165,  2,  106
DATA   74,  74,  160,  56,  189,  1,  208,  233
DATA   53,  144,  10,  201,  200,  176,  6,  74
DATA   74,  74,  170,  24,  96,  56,  96,  1
DATA   2,  4,  8,  16,  32,  64,  128
```

MEMORY/REGISTER REQUIREMENTS

This routine uses the accumulator, the .X and the .Y registers. The routine requires 3 bytes of zero page RAM storage. The routine takes up 70 bytes of memory.

WORST CASE EXECUTION TIME

100 cycles (90 microseconds in a 1.02 Mhz system)

LIMITS

As mentioned in the EXPOSITION section, there are definite limits on this routine. The problem is that a sprite takes up roughly 9 character positions on the screen when unexpanded, and 10 when expanded. The sprite to background collision could be anywhere in that 9 (or 10) character area that the sprite is covering. It is difficult to tell where the collision actually took place.

There are several solutions to the problem, none completely satisfactory. The data making up the 9 (or 10) characters could be compare with the data making up the sprite. However, this approach can lead to serious processing time, especially if the sprite does not happen to be on a character boundary.

One possible solution is to allow only certain characters could be allowed to cause a collision. Each of the 9 (or 10) characters possible must be checked on the screen for this to work. If the size of the sprite is precisely known, the number of characters to be checked might be reduced.

Another approach (which will be used in the following example) defines a 'critical area', a one character center of the sprite which we use as the key spot to inspect on the video matrix. This approach works best when the background objects on the screen are large.

There are many other approaches. The sucess of any will depend on the logic of rest of the program.

IMPORTANT NOTE: If multi-color characters are used, the problem can be reduced by making non-critical objects (objects which should not cause collisions) in 01 multi-color. The 6566/6567 chip will not report a collision between a sprite and an object of this color (bit pattern).

EXAMPLE 1

Sometimes more information than the screen matrix address of the sprite is needed. The following example uses the row and column returned by the sprite to background collision detection routine and gets the character under the top left corner of the sprite.

SOURCE LISTING

```
1000 .PAG   'CKBACK'
1010 ;
1020 ;**************************
1030 ;* BACKGROUND COLLISION
1040 ;*         ROUTINE
1050 ;**************************
1060 ;
1070 ;CONSTANTS
1080 ;
1090 SPRX   =$D000             ;VIC CHIP REGISTERS
1100 SPRY   =$D001
1110 MSIGX  =$D010
1120 SPRBAK =$D01F
1130 ;
1140 XOFF   =0                 ;SCREEN CENTERING CONSTANTS
1150 YOFF   =3
1160 ;
1170 SCREEN =$0400
1180 LLEN   =40
1190 ;
1200 XCENT  =1                 ;CENTER OF SPRITE IN ROW/COL FORM
1210 YCENT  =1
1220 ;
1230 LINE0  =SCREEN            ;SCREEN ROW CONSTANTS
1240 LINE1  =LINE0+LLEN
1250 LINE2  =LINE1+LLEN
```

```
LINE3  =LINE2+LLEN
LINE4  =LINE3+LLEN
LINE5  =LINE4+LLEN
LINE6  =LINE5+LLEN
LINE7  =LINE6+LLEN
LINE8  =LINE7+LLEN
LINE9  =LINE8+LLEN
LINE10 =LINE9+LLEN
LINE11 =LINE10+LLEN
LINE12 =LINE11+LLEN
LINE13 =LINE12+LLEN
LINE14 =LINE13+LLEN
LINE15 =LINE14+LLEN
LINE16 =LINE15+LLEN
LINE17 =LINE16+LLEN
LINE18 =LINE17+LLEN
LINE19 =LINE18+LLEN
LINE20 =LINE19+LLEN
LINE21 =LINE20+LLEN
LINE22 =LINE21+LLEN
LINE23 =LINE22+LLEN
LINE24 =LINE23+LLEN
;
;VARIABLES
;
*=$0002
;
XC      *=*+2                ;2 BYTE X COORDINATE OF SPRITE
YC      *=*+1                ;Y COORDINATE OF SPRITE
PTR     *=*+2                ;TO GET TO SCREEN
;
*=$3004
.PAG
;************************************
;* EXAMPLE OF USING THE SPRITE
;* TO BACKGROUND COLLISION ROUTINE
;*
;* THE .Y REGISTER MUST CONTAIN THE
;* SPRITE TO BE CHECKED.
;*
;* THE CHARACTER CORRESPONDING TO THE
;* SPRITE CENTER IS RETURNED IN .A
;*
;* IF THERE IS NO COLLISION THE CARRY IS
;* SET.
;************************************
;
;
;
CKBACK JSR     CORD           ;FIND SPRITE ON VIDEO MATRIX
       BCS     NOCOL          ;THE SPRITE IS OFF THE SCREEN
                              ;OR THERE IS NO COLLISION.
;
       TYA                    ;GET COLUMN NUMBER
       CLC
       ADC     #XCENT         ;ADD TO CENTER OF SPRITE (0-2)
```

```
1820         TAY
1830 ;
1840         TXA                       ;GET ROW NUMBER
1850         CLC
1860         ADC    #YCENT             ;ADD TO CENTER OF SPRITE (0-2)
1870         TAX
1880 ;
1890         LDA    SCRLOW,X           ;SET FOR INDIRECT ACCESS TO MATRIX
1900         STA    PTR
1910         LDA    SCRHI,X
1920         STA    PTR+1
1930 ;
1940         LDA    (PTR),Y            ;GET THE CHARACTER
1950         RTS
1960 ;
1970 NOCOL   SEC
1980 RTS
1990 ;
2000 ;****************************************
2010 ;* SPRITE TO BACKGROUND COLLISION ROUTINE
2020 ;*
2030 ;* ENTER THIS ROUTINE WITH THE SPRITE
2040 ;* NUMBER TO CHECK ON IN .Y
2050 ;*
2060 ;* THE COORDINATES ARE RETURNED IN .X (ROW) AND .Y (COL)
2070 ;* CARRY WILL BE CLEAR ON A GOOD RETURN
2080 ;* CARRY WILL BE SET IF SPRITE IS OFF SCREEN
2090 ;* OR IF THERE IS NO COLLISION
2100 ;****************************************
2110 ;
2120 CORD    LDA    SPRBAK
2130         AND    DOTS,Y
2140         BEQ    EXIT               ;NO COLLISION WITH THIS SPRITE
2150 ;
2160         TYA
2170         ASL    A                  ;SET POINTER FOR SPRITE REGISTERS
2180         TAX
2190 ;
2200         SEC
2210         LDA    SPRX,X             ;DO X FIRST
2220         SBC    #24+XOFF           ;SUBTRACT SCREEN OFFSET
2230         STA    XC
2240         LDA    MSIGX              ;HANDLE MSB OF SPRITE
2250         AND    DOTS,Y
2260         BEQ    SKIP1
2270 ;
2280         LDA    #1
2290 ;
2300 SKIP1   SBC    #0
2310         STA    XC+1               ;HANDLE HIGH BYTE
2320         BCC    EXIT               ;SPRITE IS OFF SCREEN
2330 ;
2340         BEQ    SKIP2              ;SPRITE IS ON LEFT SIDE OF SCREEN
2350 ;
2360         LDA    XC                 ;OFF SCREEN ?
2370         CMP    #$56
```

```
2380            BCS     EXIT                    ;YES
2390 ;
2400 SKIP2      LSR     XC+1                    ;DIVIDE BY 8
2410            LDA     XC
2420            ROR     A
2430            LSR     A
2440            LSR     A
2450 ;
2460            TAY                             ;PREPARE IT FOR RETURN
2470 ;
2480            SEC                             ;NOW DO Y COORD CONVERSION
2490            LDA     SPRY,X
2500            SBC     #50+YOFF                ;(WHERE YOFF IS FROM 0-7)
2510            BCC     EXIT                    ;SPRITE IS OFF SCREEN
2520 ;
2530            CMP     #250-50                 ;OFF THE BOTTOM ?
2540            BCS     EXIT                    ;YES
2550 ;
2560            LSR     A                       ;DIVIDE BY 8
2570            LSR     A
2580            LSR     A
2590 ;
2600            TAX                             ;PASS IT BACK IN X
2610 ;
2620            CLC                             ;SHOW GOOD EXIT
2630            RTS
2640 ;
2650 EXIT       SEC                             ;ERROR EXIT
2660            RTS
2670 ;
2680 DOTS       .BYT $01,$02,$04,$08,$10,$20,$40,$80
2690 ;
2700 SCRLOW     .BYT <LINE0, <LINE1, <LINE2, <LINE3, <LINE4, <LINE5, <LINE6
2710            .BYT <LINE7, <LINE8, <LINE9, <LINE10,<LINE11,<LINE12,<LINE13
2720            .BYT <LINE14,<LINE15,<LINE16,<LINE17,<LINE18,<LINE19,<LINE20
2730            .BYT <LINE21,<LINE22,<LINE23,<LINE24
2740 ;
2750 SCRHI      .BYT >LINE0, >LINE1, >LINE2, >LINE3, >LINE4, >LINE5, >LINE6
2760            .BYT >LINE7, >LINE8, >LINE9, >LINE10,>LINE11,>LINE12,>LINE13
2770            .BYT >LINE14,>LINE15,>LINE16,>LINE17,>LINE18,>LINE19,>LINE20
2780            .BYT >LINE21,>LINE22,>LINE23,>LINE24
2790 ;
2800 .END
```

HEX DUMP

```
.:  3004  20 22 30 80
.:  3008  17 90 18 69 01 AD 0A 10
.:  3010  69 01 AA BD 71 30 85 05
.:  3018  BD 8A 30 85 06 B1 05 60
.:  3020  38 60 89 1F D0 39 69 30
.:  3028  F0 3D 90 0A AA 38 BD 00
.:  3030  D0 E9 18 85 02 AD 10 D0
.:
```

```
.:   3038   39 69 30 F0 02 A9 01 E9
.:   3040   00 85 03 90 22 F0 06 A5
.:   3048   02 C9 56 B0 1A 46 03 A5
.:   3050   02 6A 4A 4A A8 38 BD 01
.:   3058   D0 E9 35 90 0A C9 C8 B0
.:   3060   06 4A 4A 4A AA 18 60 38
.:   3068   60 01 02 04 08 10 20 40
.:   3070   80 00 28 50 78 A0 C8 F0
.:   3078   18 40 68 90 B8 E0 08 30
.:   3080   58 80 A8 D0 F8 20 48 70
.:   3088   98 C0 04 04 04 04 04 04
.:   3090   04 05 05 05 05 05 05 06
.:   3098   06 06 06 06 06 06 07 07
.:   30A0   07 07 07
```

DATA STATEMENTS

```
DATA   32,  34,  48,  176
DATA   23,  152,  24,  105,  1,  168,  138,  24
DATA   105,  1,  170,  189,  113,  48,  133,  5
DATA   189,  138,  48,  133,  6,  177,  5,  96
DATA   56,  96,  185,  31,  208,  57,  105,  48
DATA   240,  61,  152,  10,  170,  56,  189,  0
DATA   208,  233,  24,  133,  2,  173,  16,  208
DATA   57,  105,  48,  240,  2,  169,  1,  233
DATA   0,  133,  3,  144,  34,  240,  6,  165
DATA   2,  201,  86,  176,  26,  70,  3,  165
DATA   2,  106,  74,  74,  168,  56,  189,  1
DATA   208,  233,  53,  144,  10,  201,  200,  176
DATA   6,  74,  74,  74,  170,  24,  96,  56
DATA   96,  1,  2,  4,  8,  16,  32,  64
DATA   128,  0,  40,  80,  120,  160,  200,  240
DATA   24,  64,  104,  144,  184,  224,  8,  48
DATA   88,  128,  168,  208,  248,  32,  72,  112
DATA   152,  192,  4,  4,  4,  4,  4,  4
DATA   4,  5,  5,  5,  5,  5,  5,  6
DATA   6,  6,  6,  6,  6,  6,  7,  7
DATA   7,  7,  7
```

MEMORY/REGISTER REQUIREMENTS

This routine uses the accumulator, the .X index register, the .Y index register, and 2 bytes on the stack. 5 bytes of storage on zero page are required. The routine takes up 150 bytes of memory.

WORST CASE EXECUTION TIME

150 cycles (147 microseconds on a 1.02 Mhz system)

EXAMPLE 2

Here is another example which puts the collision detection routines together with the move routines and the clock routines. This example gives an idea of how a total program can go together.

Three sprites are displayed. On the top line, left to right, are the coordinates (in hex) of sprite 2 (the red sprite), sprite 1 (the white sprite), and sprite 0 (the black sprite). Sprite to sprite collisions are displayed on the left half of the second line. The number(s) of any sprite(s) colliding with the player sprite are shown. On the right side of the second line the row and column screen matrix address (in hex) of the sprite is displayed during a sprite to background collision.

SOURCE LISTING

```
.PAGE  'ADEC'
;
;
*=$0002
;
;VARIABLES
OSH      *=*+1            ;OFFSET HIGH
OSL      *=*+1            ;OFFSET LOW
MODH     *=*+1            ;MODULUS HIGH
MODL     *=*+1            ;MODULUS LOW
TEMX     *=*+1            ;TEMP .X
TA       *=*+1            ;TEMP .A
SMOB     *=*+8            ;AUXILIARY SPRITE MOB BYTES
CLKF     *=*+1            ;FRAME CLOCK
CLKM     *=*+1            ;MINUTES CLOCK
CLKS     *=*+1            ;SECONDS CLOCK
```

```
1160 EFC     *=*+0          ;EVENT FRAME COUNTER
1170 FPS     *=*+1          ;FRAMES PER SECOND (T.V. STANDARD)
1180 MOD     *=*+1          ;MODULUS
1190 RCOMP   *=*+1          ;RASTER COMPARE VALUE
1200 DX      *=*+1          ;JOYSTICK X DIRECTION OFFSET
1210 DY      *=*+1          ;JOYSTICK Y DIRECTION OFFSET
1220 SNUM    *=*+1          ;SPRITE NUMBER TO MOVE
1230 ;
1240 COUNT   *=*+1          ;NUMBER OF VALID COLLISIONS
1250 PLAYER  *=*+1          ;SPRITE TO CHECK ON (PLAYER SPRITE)
1260 PXMAX   *=*+1          ;MAXIMUM SIZE OF PLAYER IN X DIRECTION
1270 PYMAX   *=*+1          ;MAXIMUM SIZE OF PLAYER IN Y DIRECTION
1280 COLIDE  *=*+1          ;TEMPORARY STORAGE FOR COLLISION REGISTER
1290 TX      *=*+2          ;TEMPORARY SPRITE X POS STORAGE
1300 TY      *=*+1          ;TEMPORARY SPRITE Y POS STORAGE
1310 ;
1320 PLX     *=*+2          ;PLAYER SPRITE X POSITION
1330 PLY     *=*+1          ;PLAYER SPRITE Y POSITION
1340 TEMP    *=*+1
1350 RESULT  *=*+7          ;SPRITE NUMBERS OF VALUE COLLISIONS
1360 ;
1370 XC      *=*+2          ;2 BYTE X COORDINATE OF SPRITE
1380 YC      *=*+1          ;Y COORDINATE OF SPRITE
1390 ;
1400 ;CONSTANTS
1410 ;
1420 SCREEN  =$0400
1430 SPRPTR  =$07F8         ;SPRITE DATA POINTER
1440 SPRX    =$D000         ;VIC REGISTER: SPRITE #0 X POSITION
1450 SPRY    =$D001         ;VIC REGISTER: SPRITE #0 Y POSITION
1460 MSIGX   =$D010         ;VIC REGISTER: SPRITE MSB BITS
1470 RASHGH  =$D011         ;VIC REGISTER: RASTER MSB
1480 RASTER  =$D012         ;VIC REGISTER: RASTER LSB'S
1490 SPENA   =$D015         ;VIC REGISTER: SPRITE ENABLE
1500 YXPAND  =$D017         ;SPRITE Y EXPAND
1510 XXPAND  =$D01D         ;VIC REGISTER: SPRITE X-EXPANSION
1520 SPRSPR  =$D01E
1530 SPRBAK  =$D01F
1540 SCOLOR  =$D027
1550 ;
1560 XOFF    =0             ;SCREEN CENTERING CONSTANTS
1570 YOFF    =3
1580 ;
1590 .END
1600 .PAGE  'EXAMPLE'
1610 ;
1620         *=$3000
1630 ;
1640 EXAM    SEI            ;DISABLE IRQS
1650         LDA #50        ;ASSUME PAL UNLESS FOUND OTHERWISE
1660         STA FPS        ;  (PAL IS 50 FRAMES PER SEC)
1670         LDX #$01       ;SET UP PAL MODULUS MSB
1680         LDY #$F8       ;SET UP PAL MODULUS LSB
1690 EX0     LDA RASHGH     ;LOOK AT RASTER MSB IS IT A 1 ?
1700         BPL EX0        ;IF 0 THEN RASTER < 256. SO LOOK AGAIN
1710 EX1     LDA #$08       ;RASTER > 255 ...BUT...
```

```
1720          CMP RASTER        ;IS IT GREATER THAN 264 ?
1730          BCC EX2           ;IF YES THEN BRANCH. TV STD = PAL !!
1740          LDA RASHGH        ;IF NO THEN CHECK FOR MOD OF RASTER=1
1750          BMI EX1           ;IF YES THEN GOTO EX1
1760          LDX #$02          ;SET UP NTSC MODULUS MSB. TV STD = NTSC
1770          LDY #$00          ;SET UP NTSC MODULUS LSB
1780          LDA #60           ;SET UP NTSC FRAMES PER SECOND
1790          STA FPS           ;  (NTSC IS 60 FRAMES PER SECOND)
1800 EX2      STX MODH          ;STORE IN MODULUS FOR FUTURE USE
1810          STY MODL
1820 ;
1830          LDY #0            ;COLOR SCREEN WHITE
1840          LDA #1            ;WHITE
1850 EX02     STA $D800,Y
1860          STA $D800+256,Y
1870          STA $D800+512,Y
1880          STA $D800+768,Y
1890          DEY
1900          BNE EX02
1910 ;
1920 ; INITIALIZE SPRITE VALUES
1930 ;
1940          LDA #$7F          ;SET SOME SPRITES TO 127
1950          STA SPRY
1960          STA SPRX+2
1970          ;
1980          LDA #50           ;SET OTHERS TO 50
1990          STA SPRX
2000          STA SPRY+2
2010 ;
2020          LDA #100          ;STILL OTHERS TO 100
2030          STA SPRX+4
2040          STA SPRY+4
2050 ;
2060          LDX #2
2070 XAG      LDA #$00          ;CLEAR OUT SMSBS & MS1GX
2080          STA SMSB,X        ;& SET SPRITE COLORS
2090          TXA
2100          STA SCOLOR,X
2110          DEX
2120          BPL XAG
2130 ;
2140          STA MS1GX
2150          STA SNUM          ;START MOVING SPRITE #0
2160 ;
2170          LDA #$00          ;SET POINTERS TO SPRITE PICTURE
2180          STA SPRPTR
2190          STA SPRPTR+1
2200          STA SPRPTR+2
2210 ;
2220          LDA #7            ;ENABLE AND EXPAND SPRITES 0,1,2
2230          STA XXPAND
2240          STA YXPAND
2250          STA SPENA
2260 ;
2270          LDX #62           ;SET SPRITE
```

```
2280 EX3     LDA SDATA,X
2290         STA $2000,X        ;NOTE:- WHEN SPRITE POINTER=128
2300 ;
2310         DEX                ;SPRITE ADDRESS=$2000
2320         BPL EX3
2330 ;
2340         LDA #44            ;SET SPRITE SIZE FOR COLLISION DETECTION
2350         STA PXMAX
2360         LDA #30
2370         STA PYMAX
2380 ;
2390         JSR VTAMX          ;TRANSFER MSIGX BITS TO SMSB BYTES
2400         LDA #$FA           ;SET RASTER COMPARE TO JUST OFF
2410         STA RCOMP          ;THE BOTTOM OF VISIBLE VIEWING AREA
2420 ;
2430 EX4     JSR DJRR           ;READ THE JOYSTICK
2440         BCS EX5            ;IF C=0 THEN FIRE BUTTON PUSHED CHANGE SPR
2450 ;
2460         LDA SNUM           ;NOW DO OTHER MOVING SPRITE
2470         EOR #$01
2480         STA SNUM
2490 WEX     JSR DJRR           ;WAIT UNTIL BUTTON IS NO LONGER PRESSED
2500         BCC WEX
2510         BCS EX4
2520 ;
2530 EX5     LDA SNUM           ;LOAD .A WITH SPRITE NO.
2540         LDX DX             ;LOAD .X WITH JOYSTICK X DIRECTION OFFSET
2550         JSR UNIMVX         ;MOVE SPRITE MODULO (504 PAL OR 512 NTSC)
2560         JSR ATVMX          ;TRANSFER SMSB BYTES TO MSIGX BITS
2570         LDA SNUM           ;LOAD .A WITH SPRITE NO.
2580         LDY DY             ;LOAD .Y WITH JOYSTICK Y DIRECTION OFFSET
2590         JSR MVSY           ;MOVE THE SPRITE IN Y DIRECTION
2600 ;
2610 ;***    OUTPUT SPRITE COORDS & COLLISION STATUS TO SCREEN
2620 ;
2630         LDX #2
2640         LDY #5             ;CENTER THE OUTPUT LINE
2650         ;
2660 OAG     LDA SMSB,X         ;CONVERT SPRITE X COORDINATE TO ASCII HEX
2670         JSR BTH            ;CONVERT BINARY TO SCREEN ASCII
2680         TXA                ;MULT BY 2 FOR A SECOND
2690         ASL A
2700         TAX
2710         LDA SPRX,X         ;DO FOR X LSB'S...
2720         JSR BTH
2730 ;
2740         LDA SPRY,X         ;...AND Y
2750         INY                ;SKIP A SPACE BETWEEN X AND Y COORDS
2760         INY
2770         JSR BTH
2780         INY                ;SKIP BETWEEN SPRITES
2790         INY
2800         INY
2810 ;
2820         TXA                ;NOW SET IT BACK
2830         LSR A
```

```
2840         TAX
2850 ;
2860         DEX
2870         BPL OAG
2880 ;
2890         JSR CLOCK       ;SYNCHRONIZE PROGRAM WITH RASTER
2900 ;
2910         LDX #39         ;BLANK SECOND LINE
2920         LDA #$20
2930 BAG     STA $0428,X
2940         DEX
2950         BPL BAG
2960 ;
2970         LDX SNUM        ;NOW DO COLLISION CHECK WITH MAIN SPRITE
2980         JSR BANG
2990         LDX COUNT       ;ANY COLLISIONS ?
3000         BEQ EX6         ;NO, NOT THIS TIME
3010         ;
3020         LDY #42         ;COLLISIONS ARE SHOWN ON SECOND LINE
3030 OOAG    LDA RESULT-1,X
3040         JSR BTH
3050         INY             ;SKIP A SPACE BETWEEN COLLISIONS
3060         INY
3070         DEX
3080         BNE OOAG
3090 ;
3100 EX6     LDY SNUM        ;NOW SHOW BACKGROUND COLLISIONS
3110         JSR ECORD
3120         BCS EX4         ;SPRITE IS OFF SCREEN OR NO COLLISION
3130 ;
3140         TYA             ;GET X COORD
3150         LDY #72         ;AND OUTPUT IT
3160         JSR BTH
3170 ;
3180         INY             ;SKIP A SPACE
3190         INY
3200         TXA             ;GET Y COORD
3210         JSR BTH
3220 ;
3230         JMP EX4
3240 ;
3250 ;***    CONVERT BYTE TO TWO SCREEN HEX CHARACTERS
3260 ;
3270 BTH     PHA
3280         AND #$0F
3290         JSR CONV
3300         STA $0401,Y
3310         PLA
3320         LSR A
3330         LSR A
3340         LSR A
3350         LSR A
3360         JSR CONV
3370         STA $0400,Y
3380         INY             ;MOVE TO NEXT OUTPUT POSITION
3390         INY
```

```
3400         RTS
3410 ;
3420 ;***      CONVERT NYBBLE TO SCREEN CHARACTER HEX
3430 ;
3440 CONV    CMP #$0A
3450         BCC CONV1
3460         SBC #$09
3470         RTS
3480 CONV1   ORA #$30
3490         RTS
3500 ;
3510 ;****************************************
3520 ;*       UNIVERSAL MOVE SPRITE IN X DIRECTION
3530 ;*
3540 ;*    A REG=SPRITE NO.
3550 ;*    X REG=OFFSET (2'S COMPLEMENT FORM)
3560 ;*
3570 ;****************************************
3580 ;
3590 UNIMVX  STX TEMX        ;PROTECT X
3600         STX OSL         ;SET UP OFFSET LOW
3610         TAX             ;.X=.A
3620         ASL A           ;.A=2*A
3630         TAY             ;.Y=A
3640         LDA #$00        ;CLEAR OFFSET HIGH
3650         STA OSH
3660         CLC
3670         LDA OSL         ;CHECK FOR NEGATIVE OFFSET
3680         BPL UMX0        ;IF POSITIVE THEN BRANCH
3690         EOR #$FF        ;PERFORM 2'S COMPLEMENT OPERATION
3700         ADC #$01
3710         STA OSL         ;PUT RESULTS IN OSL AND THEN
3720         SEC             ;FORM THE MODULAR COMPLEMENT OF
3730         LDA MODL        ;THE OFFSET BY SUBTRACTING IT
3740         SBC OSL         ;FROM THE MODULUS
3750         STA OSL
3760         LDA MODH        ;PAL 504, NTSC 512
3770         SBC OSH
3780         STA OSH
3790         CLC
3800 UMX0    LDA SPRX,Y      ;ADD OFFSET TO SPRITE X
3810         ADC OSL         ;[SMSB+X,SPRX+Y]=...
3820         STA SPRX,Y      ;...[SMSB+X,SPRX+Y]+[OSH,OSL]
3830         LDA SMSB,X
3840         ADC OSH
3850         STA SMSB,X
3860         SEC             ;IS THE SUM >= MODULUS
3870         LDA SPRX,Y      ;CHECK BY SUBTRACTING
3880         SBC MODL
3890         STA TA          ;CATCH FOR LATER USE
3900         LDA SMSB,X
3910         SBC MODH
3920         BCC UMX1        ;IF SUM < MOD THEN EXIT
3930         STA SMSB,X      ;OTHERWISE CORRECT X COORD
3940         LDA TA
3950         STA SPRX,Y
```

```
UMX1    TYA              ;RESTORE A REG
        LSR A
        LDX TEMX          ;RESTORE X REG
        RTS
;
;****************************************
;* TRANSFER SPRITE MSB BYTES TO MSIGX BITS
;****************************************
;
ATVMX   LDX #$07
ATV0    LDA SMSB,X
        LSR A
        ROL MSIGX
        DEX
        BPL ATV0
        RTS
;
;****************************************
;* TRANSFER SPRITE MSIGX BITS TO MSB BYTES
;****************************************
;
VTAMX   LDX #$07
        LDA MSIGX
VTA0    LSR SMSB,X
        ASL A
        ROL SMSB,X
        DEX
        BPL VTA0
        RTS
;
;******************************
;* MOVE SPRITE IN Y DIRECTIONS
;******************************
;
MVSY    ASL A            ;AT START .A SHOULD BE LOADED WITH SPR#
        TAX              ;.X=2*SPR#
        TYA              ;OFFSET MOVED INTO .A
        ADC SPRY,X       ;OFFSET IS ADDED TO SPR Y POSITION
        STA SPRY,X       ;SUM IS NEW Y POSITION
        CMP #$1D         ;IF Y<$1D THEN C=0
        BCC MVSY0        ;  (AND BRANCH TO EXIT)
        LDA #$F9         ;  ELSE IS Y>$F9? (LAST Y ON SCREEN)
        CMP SPRY,X       ;CARRY IS UPDATED ACCORDINGLY
MVSY0   RTS              ;EXIT
;
;**************************************
;* JOYSTICK/FIRE BUTTON READ
;* IF CARRY =0 THEN FIRE BUTTON PRESSED
;**************************************
;
DJRR    LDA $DC00        ;  (GET INPUT FROM PORT A ONLY)
DJRRB   LDY #0           ;READ AND DECODE THE JOYSTICK/FIRE
        LDX #0           ;BUTTON INPUT DATA IN .A; THE 5 LSB'S
        LSR A            ;CONTAIN THE SWITCH CLOSURE INFORMATION.
        BCS DJR0         ;IF A SWITCH IS CLOSED THEN IT PRO-
        DEY              ;DUCES A 0 BIT. IF A SWITCH IS OPEN
```

```
4520 DJR0    LSR A                ;THEN IT PRODUCES A 1 BIT. THE JOY-
4530         BCS DJR1             ;STICK DIRECTIONS ARE RIGHT, LEFT,
4540         INY                  ;FORWARD, BACKWARD. B3=RIGHT, B2=LEFT,
4550 DJR1    LSR A                ;B1=BACKWARD, B0=FORWARD AND B4=FIRE
4560         BCS DJR2             ;BUTTON. AT RTS TIME DX AND DY CONTAIN
4570         DEX                  ;2'S COMPLEMENT DIRECTION #'S, I.E.
4580 DJR2    LSR A                ;$FF=-1, $00=0, $01=1. DX=-1 (MOVE
4590         BCS DJR3             ;RIGHT), DX=1 (MOVE LEFT), DX=0 (NO X
4600         INX                  ;CHANGE). DY=-1 (MOVE UP SCREEN), DY=1
4610 DJR3    LSR A                ;(MOVE DOWN SCREEN), DY=0 (NO Y CHANGE).
4620         STX DX               ;THE FORWARD JOYSTICK POSITION CORRESPONDS
4630         STY DY               ;TO MOVE UP THE SCREEN AND THE BACKWARD
4640         RTS                  ;POSITION TO MOVE DOWN SCREEN.
4650 ;
4660 ;
4670 ;****************************************
4680 ;* CLOCK SUBROUTINE AND SUPPORT SUBROUTINES
4690 ;****************************************
4700 ;
4710 CLOCK   JSR NWAIT            ;WAIT UNTIL START OF NEXT FRAME
4720         LDX #$07             ;INCREMENT THE 8 EVENT FRAME
4730 CLK0    INC EFC,X            ;COUNTERS MODULO 256
4740         DEX
4750         BPL CLK0
4760         LDA FPS              ;FRAMES PER SECOND VALUE
4770         STA MOD              ;(50-PAL, 60-NTSC)
4780         LDX CLKF
4790         JSR MODINC           ;X = X+1 MOD (FPS)
4800         STX CLKF             ;UPDATE FRAME CLOCK
4810         BNE CLK1             ;NO MODULUS CROSSING THEREFORE EXIT
4820         LDA #60              ;IF MODULUS CROSSING THEN...
4830         STA MOD              ;SET 'MOD' TO 60 FOR 60 SEC/MIN
4840         LDX CLKS
4850         JSR MODINC           ;X = X+1 MOD (60)
4860         STX CLKS             ;UPDATE SECONDS CLOCK
4870         BNE CLK1             ;NO MODULUS CROSSING THEREFORE EXIT
4880         INC CLKM             ;OTHERWISE UPDATE MINUTE CLOCK
4890 CLK1    RTS
4900 ;
4910 ;*********************************
4920 ;* INC .X MODULO THE VALUE IN 'MOD'.
4930 ;*********************************
4940 ;
4950 MODINC INX
4960         CPX MOD              ;CHECK FOR MODULUS CROSSING
4970         BCC MODIN1           ;IF X < MOD THEN EXIT
4980         LDX #$00             ;OTHERWISE X = 0
4990 MODIN1 RTS
5000 ;
5010 ;************************************
5020 ;* WAIT UNTIL THE RASTER VALUE MISMATCHES
5030 ;* THE CHOSEN COMPARE VALUE AND THEN WAIT
5040 ;* UNTIL THE RASTER VALUE MATCHES THE
5050 ;* COMPARE VALUE.
5060 ;************************************
5070 ;
```

```
5060 NWAIT   LDA RASTER       ;CHECK RASTER
5070         CMP RCOMP        ;FOR MISMATCH
5100         BEQ NWAIT        ;IF MATCH THEN CHECK
5110 WAIT    LDA RASTER       ;RASTER AGAIN
5120         CMP RCOMP        ;FOR MATCH.
5130         BNE WAIT         ;IF MISMATCH THEN CHECK AGAIN
5140         RTS
5150 ;
5160 .END
```

HEX DUMP

```
.: 3000   78 A9 32 85 1B A2 01 A0
.: 3008   F8 AD 11 D0 10 FB A9 08
.: 3010   CD 12 D0 90 0D AD 11 D0
.: 3018   30 F4 A2 02 A0 00 A9 3C
.: 3020   85 1B 86 04 84 05 A0 00
.: 3028   A9 01 99 00 D8 99 00 D9
.: 3030   99 00 DA 99 00 DB C8 D0
.: 3038   F1 A9 7F 8D 01 D0 8D 02
.: 3040   D0 A9 32 8D 00 D0 8D 03
.: 3048   D0 A9 64 8D 04 D0 8D 05
.: 3050   D0 A2 02 A9 00 95 08 CA
.: 3058   9D 27 D0 CA 10 F5 8D 10
.: 3060   D0 85 20 A9 80 8D F8 07
.: 3068   8D F9 07 8D FA 07 A9 07
.: 3070   8D 1D D0 8D 17 D0 8D 15
.: 3078   D0 A2 3E BD D6 32 9D 00
.: 3080   20 CA 10 F7 A9 2C 85 23
.: 3088   A9 1E 85 24 20 8A 31 A9
.: 3090   FA 85 1D 20 AD 31 B0 0D
.: 3098   A5 20 49 01 85 20 20 A8
.: 30A0   31 90 FB B0 EC A5 20 A6
.: 30A8   1E 20 32 31 20 7E 31 A5
.: 30B0   20 A4 1F 20 98 31 A2 02
.: 30B8   A0 05 B5 08 20 11 31 8A
.: 30C0   0A AA BD 00 D0 20 11 31
.: 30C8   BD 01 D0 C8 C8 20 11 31
.: 30D0   C8 C8 C8 8A 4A AA CA 10
.: 30D8   E1 20 C8 31 A2 27 A9 20
.: 30E0   9D 20 04 CA 10 FA A6 20
.: 30E8   20 06 32 A6 21 F0 0C A0
.: 30F0   2A B5 2C 20 11 31 C8 C8
.: 30F8   CA D0 F6 A4 20 20 8F 32
.: 3100   B0 91 90 A0 40 20 11 31
.: 3108   C8 C8 8A 20 11 31 4C 93
.: 3110   30 48 29 0F 20 20 31 99
.: 3118   01 04 68 4A 4A 4A 4A 20
.: 3120   20 31 99 00 04 C8 C8 60
.: 3128   C9 0A 90 03 E9 09 60 09
.: 3130   30 60 86 06 86 03 AA 0A
.: 3138   A8 A9 00 85 02 18 A5 03
.: 3140   10 14 49 FF 69 01 85 03
.: 3148   38 A5 05 E5 03 85 03 A5
```

```
.: 3150   04 E5 02 85 02 18 B9 00
.: 3158   D0 65 03 99 00 D0 B5 08
.: 3160   65 02 95 08 38 B9 00 D0
.: 3168   E5 05 85 07 B5 08 E5 04
.: 3170   90 07 95 08 A5 07 99 00
.: 3178   D0 98 4A A6 06 60 A2 07
.: 3180   B5 08 4A 2E 10 D0 CA 10
.: 3188   F7 60 A2 07 AD 10 D0 56
.: 3190   08 0A 36 08 CA 10 F8 60
.: 3198   0A AA 98 7D 01 D0 9D 01
.: 31A0   D0 C9 1D 90 05 A9 F9 DD
.: 31A8   01 D0 60 AD 00 DC A0 00
.: 31B0   A2 00 4A B0 01 88 4A B0
.: 31B8   01 C8 4A B0 01 CA 4A B0
.: 31C0   01 E8 4A 86 1E 84 1F 60
.: 31C8   20 F7 31 A2 07 F6 13 CA
.: 31D0   10 FB A5 1B 85 1C A6 10
.: 31D8   20 EF 31 86 10 D0 0F A9
.: 31E0   38 85 1C A6 12 20 EF 31
.: 31E8   86 12 D0 02 E6 11 60 E8
.: 31F0   E4 1C 90 02 A2 00 60 AD
.: 31F8   12 D0 C5 1D F0 F9 AD 12
.: 3200   D0 C5 1D D0 F9 60 A9 00
.: 3208   85 21 AD 1E D0 85 25 86
.: 3210   22 BD 87 32 25 25 F0 54
.: 3218   20 6D 32 85 29 A5 27 85
.: 3220   2A A5 28 85 2B A2 07 E4
.: 3228   22 F0 3E BD 87 32 25 25
.: 3230   F0 37 20 6D 32 38 E5 29
.: 3238   A8 A5 27 E5 2A B0 0E 85
.: 3240   2C 98 49 FF 69 01 A8 A5
.: 3248   2C 49 FF 69 00 D0 1A 98
.: 3250   C5 23 B0 15 38 A5 2B E5
.: 3258   28 B0 04 49 FF 69 01 C5
.: 3260   24 B0 06 E6 21 A4 21 96
.: 3268   2C CA 10 B8 60 0A 0A A8
.: 3270   B9 01 D0 85 28 AD 10 D0
.: 3278   3D 87 32 F0 02 A9 01 85
.: 3280   27 B9 00 D0 85 26 60 01
.: 3288   02 04 08 10 20 40 80 AD
.: 3290   1F D0 39 87 32 F0 3D 98
.: 3298   0A AA 38 BD 00 D0 E9 18
.: 32A0   85 34 AD 10 D0 39 87 32
.: 32A8   F0 02 A9 01 E9 00 85 35
.: 32B0   90 22 F0 06 A5 34 C9 56
.: 32B8   B0 1A 46 35 A5 34 6A 4A
.: 32C0   4A A8 38 BD 01 D0 E9 35
.: 32C8   90 0A C9 C8 B0 06 4A 4A
.: 32D0   4A AA 18 60 38 60 00 30
.: 32D8   00 00 5C 00 00 7E 00 01
.: 32E0   1F 00 02 1F 80 04 3F C0
.: 32E8   08 DF E0 13 1F F0 2C 1F
.: 32F0   F8 70 1F FC 60 1F FC 30
.: 32F8   1F F8 0C 1F E0 03 1F 80
.: 3300   00 DE 00 00 38 00 00 00
.: 3308   00 00 00 00 00 00 00 00
```

```
.: 3310   00 00 00 00 00
```

DATA STATEMENTS

```
DATA   120,  169,  50,  133,  27,  162,  1,  160
DATA   248,  173,  17,  208,  16,  251,  169,  8
DATA   205,  18,  200,  144,  13,  173,  17,  208
DATA   48,  244,  162,  2,  160,  0,  169,  60
DATA   133,  27,  134,  4,  132,  5,  160,  0
DATA   169,  1,  153,  0,  216,  153,  0,  217
DATA   153,  0,  218,  153,  0,  219,  136,  208
DATA   241,  169,  127,  141,  1,  208,  141,  2
DATA   208,  169,  50,  141,  0,  208,  141,  3
DATA   208,  169,  100,  141,  4,  208,  141,  5
DATA   208,  162,  2,  169,  0,  149,  0,  138
DATA   157,  39,  208,  202,  16,  245,  141,  16
DATA   208,  133,  32,  169,  128,  141,  248,  7
DATA   141,  249,  7,  141,  250,  7,  169,  7
DATA   141,  29,  208,  141,  23,  208,  141,  21
DATA   208,  162,  62,  189,  214,  50,  157,  0
DATA   32,  202,  16,  247,  169,  44,  133,  35
DATA   169,  30,  133,  36,  32,  138,  49,  169
DATA   250,  133,  29,  32,  171,  49,  176,  13
DATA   165,  32,  73,  1,  133,  32,  32,  171
DATA   49,  144,  251,  176,  230,  165,  32,  166
DATA   30,  32,  50,  49,  32,  126,  49,  165
DATA   32,  164,  31,  32,  152,  49,  162,  2
DATA   160,  5,  181,  8,  32,  17,  49,  138
DATA   10,  170,  189,  0,  208,  32,  17,  49
DATA   189,  1,  208,  200,  200,  32,  17,  49
DATA   200,  200,  200,  138,  74,  170,  202,  16
DATA   225,  32,  208,  49,  162,  39,  169,  32
DATA   157,  40,  4,  202,  16,  250,  166,  32
DATA   32,  6,  50,  166,  33,  240,  12,  160
DATA   42,  181,  44,  32,  17,  49,  200,  200
DATA   202,  208,  246,  164,  32,  32,  143,  50
DATA   176,  145,  152,  160,  72,  32,  17,  49
DATA   200,  200,  138,  32,  17,  49,  76,  147
DATA   48,  72,  41,  15,  32,  40,  49,  153
DATA   1,  4,  104,  74,  74,  74,  74,  32
DATA   40,  49,  153,  0,  4,  200,  200,  96
DATA   201,  10,  144,  3,  233,  9,  96,  9
DATA   48,  96,  134,  6,  134,  3,  170,  10
DATA   168,  169,  0,  133,  2,  24,  165,  3
DATA   16,  20,  73,  255,  105,  1,  133,  3
DATA   56,  165,  5,  229,  3,  133,  3,  165
DATA   4,  229,  2,  133,  2,  24,  185,  0
DATA   208,  101,  3,  153,  0,  208,  181,  8
DATA   101,  2,  149,  8,  56,  185,  0,  208
DATA   229,  5,  133,  7,  181,  8,  229,  4
DATA   144,  7,  149,  8,  165,  7,  153,  0
DATA
```

```
DATA   208, 152, 74, 166, 6, 96, 162, 7
DATA   181, 8, 74, 46, 16, 208, 202, 16
DATA   247, 96, 162, 7, 173, 16, 208, 86
DATA   8, 10, 54, 8, 202, 16, 240, 96
DATA   10, 170, 152, 125, 1, 208, 157, 1
DATA   208, 201, 29, 144, 5, 169, 249, 221
DATA   1, 208, 96, 173, 0, 220, 160, 0
DATA   162, 0, 74, 176, 1, 136, 74, 176
DATA   1, 200, 74, 176, 1, 202, 74, 176
DATA   1, 232, 74, 134, 30, 132, 31, 96
DATA   32, 247, 49, 162, 7, 246, 19, 202
DATA   16, 251, 165, 27, 133, 28, 166, 16
DATA   32, 239, 49, 134, 16, 208, 15, 169
DATA   60, 133, 28, 166, 18, 32, 239, 49
DATA   134, 18, 208, 2, 230, 17, 96, 232
DATA   228, 28, 144, 2, 162, 0, 96, 173
DATA   18, 208, 197, 29, 240, 249, 173, 18
DATA   208, 197, 29, 208, 249, 96, 169, 0
DATA   133, 33, 173, 30, 208, 133, 37, 134
DATA   34, 189, 135, 50, 37, 37, 240, 84
DATA   32, 189, 50, 133, 41, 165, 39, 133
DATA   42, 165, 40, 133, 43, 162, 7, 228
DATA   34, 240, 62, 189, 135, 50, 37, 37
DATA   240, 55, 32, 189, 50, 56, 229, 41
DATA   168, 165, 39, 229, 42, 176, 14, 133
DATA   44, 152, 73, 255, 105, 1, 168, 165
DATA   44, 73, 255, 105, 0, 208, 26, 152
DATA   197, 35, 176, 21, 56, 165, 40, 229
DATA   43, 176, 4, 73, 255, 105, 1, 197
DATA   36, 176, 6, 230, 33, 164, 33, 150
DATA   44, 202, 16, 187, 96, 138, 10, 168
DATA   185, 1, 208, 133, 40, 173, 16, 208
DATA   61, 135, 50, 240, 2, 169, 1, 133
DATA   39, 185, 0, 208, 133, 38, 96, 1
DATA   2, 4, 8, 16, 32, 64, 128, 173
DATA   31, 208, 57, 135, 50, 240, 61, 152
DATA   10, 170, 56, 189, 0, 208, 233, 24
DATA   133, 52, 173, 16, 208, 57, 135, 50
DATA   240, 2, 169, 1, 233, 0, 133, 53
DATA   144, 34, 240, 6, 165, 52, 201, 86
DATA   176, 26, 70, 53, 165, 52, 106, 74
DATA   74, 168, 56, 189, 1, 208, 233, 53
DATA   144, 10, 201, 200, 176, 6, 74, 74
DATA   74, 170, 24, 96, 56, 96, 0, 56
DATA   0, 0, 72, 0, 0, 158, 0, 1
DATA   31, 0, 2, 31, 128, 4, 63, 192
DATA   8, 223, 224, 19, 31, 240, 44, 31
DATA   240, 112, 31, 252, 96, 31, 252, 48
DATA   31, 248, 12, 31, 224, 3, 31, 128
DATA   0, 222, 0, 0, 56, 0, 0, 0
DATA   0, 0, 0, 0, 0, 0, 0, 0
DATA   0, 0, 0, 0, 0
```

SOFTWARE APPLICATION

NOTE 1005.a

Authors: Eric Cotton and Andy Finkel

Subject: Raster Scan - Multiple Sprite Interrupt Routine

Television Standard: NTSC and PAL

I. ABSTRACT: The purpose of this application note is to supply a method for displaying up to 16 sprites on the screen simultaneously. When a program uses one to eight sprites at a time, the VIC II sprite registers are usually updated once per frame while the raster is not scanning the visible viewing area. If, however, immediately after a sprite is drawn on the screen we change its position to a location following the raster, the sprite will be drawn once again at the new location. For example, suppose we position a sprite on the screen at x=200, y=70. If, after the raster moves past the bottom of the sprite, we change the y coordinate to 150 the raster will draw the sprite again at x=200, y=150. For a brief instant the sprite will be displayed on two parts of the screen simultaneously. By applying this technique to the other sprites we find that we can display 16 sprites (or more, if we repeat the process) at a time.

II. EXPOSITION: The IRQ service routine (IRQSVC)

presented in this application note allows the user to display 16 sprites on the screen at a time (under program control). Instead of writing sprite information directly to the VIC, the user should write to two pseudo VIC chips located on zero page. Each pseudo chip controls eight sprites (set1 and set2). Set 1 sprites should be limited to y positions in the top half of the screen and set 2 sprites should be limited to y positions in the bottom half. Each set is alternately transfered to the real VIC chip by the IRQ routine.

Interrupts are generated when the raster is at the middle of the screen (raster=$98) and when it is just past the visible viewing area (raster=$FA). The first transfers data from the bottom eight sprites (set 2) to the VIC chip, while the second transfers the data from the top eight (set 1).

Included is a set-up routine which: (1) disables interrupts from the 6526 CIA, (2) loads the IRQ vector at $0314-$0315 with the address of the new routine, (3) disables all VIC II interrupts except for raster-compare, and (4) sets the first interrupt for raster line $FA.

A. Pseudo VIC Chip Register Map: The following map shows the address, name, set, and corresponding VIC II chip location (where applicable) for each pseudo register. All numbers are in hexadecimal.

ADDR	NAME	SET	VIC
03	S1XLSB+0	1	00
04	S1XLSB+1	1	02
05	S1XLSB+2	1	04
06	S1XLSB+3	1	06
07	S1XLSB+4	1	08
08	S1XLSB+5	1	0A
09	S1XLSB+6	1	0C
0A	S1XLSB+7	1	0E
0B	S2XLSB+0	2	00
0C	S2XLSB+1	2	02
0D	S2XLSB+2	2	04
0E	S2XLSB+3	2	06
0F	S2XLSB+4	2	08
10	S2XLSB+5	2	0A
11	S2XLSB+6	2	0C
12	S2XLSB+7	2	0E
13	S1XMSB	1	10
14	S2XMSB	2	10
15	S1SPRY+0	1	01
16	S1SPRY+1	1	03
17	S1SPRY+2	1	05
18	S1SPRY+3	1	07
19	S1SPRY+4	1	09
1A	S1SPRY+5	1	0B
1B	S1SPRY+6	1	0D
1C	S1SPRY+7	1	0F
1D	S2SPRY+0	2	01
1E	S2SPRY+1	2	01
1F	S2SPRY+2	2	01
20	S2SPRY+3	2	01
21	S2SPRY+4	2	01
22	S2SPRY+5	2	01
23	S2SPRY+6	2	01
24	S2SPRY+7	2	01
25	S1SPTR+0	1	—
26	S1SPTR+1	1	—
27	S1SPTR+2	1	—
28	S1SPTR+3	1	—

ADDR	NAME	SET	VIC
29	S2SPTR+4	1	—
2A	S2SPTR+5	1	—
2B	S2SPTR+6	1	—
2C	S1SPTR+7	1	—
2D	S2SPTR+0	2	—
2E	S2SPTR+1	2	—
2F	S2SPTR+2	2	—
30	S2SPTR+3	2	—
31	S2SPTR+4	2	—
32	S2SPTR+5	2	—
33	S2SPTR+6	2	—
34	S2SPTR+7	2	—
35	S1COLR+0	1	27
36	S1COLR+1	1	28
37	S1COLR+2	1	29
38	S1COLR+3	1	2A
39	S1COLR+4	1	2B
3A	S1COLR+5	1	2C
3B	S1COLR+6	1	2D
3C	S1COLR+7	1	2E
3D	S2COLR+0	2	27
3E	S2COLR+1	2	28
3F	S2COLR+2	2	29
40	S2COLR+3	2	2A
41	S2COLR+4	2	2B
42	S2COLR+5	2	2C
43	S2COLR+6	2	2D
44	S2COLR+7	2	2E
45	S1XPND	1	1D
46	S2XPND	2	1D
47	S1YPND	1	17
48	S2YPND	2	17
49	S1SCOL	1	1E
4A	S2SCOL	2	1E
4B	S1SDCL	1	1B
4C	S2SDCL	2	1B

B. Program Listings:

1. Source:

```
1000 .PAGE 'IRQSVC'
1010 ;
1020 ;IRQSVC: THE IRQSVC ROUTINE ENABLES THE USE
1030 ;OF 16 SPRITES DIVIDED INTO TWO SETS. SET1
1040 ;IS LIMITED TO POSITIONS ON THE TOP HALF OF
1050 ;THE VISIBLE VIEWING AREA, SET2 IS LIMITED
1060 ;TO THE BOTTOM HALF.    THE SET UP ROUTINE
1070 ;(SETUP) SHOULD BE EXECUTED ONCE AT THE START
1080 ;
1090 *=$02                    ;OR ELSEWHERE ON ZERO PAGE
1100 ;VARIABLES
1110 HOLDX  *=*+1             ;TEMPORARY STORAGE FOR X REGISTER
1120 ;
1130 ; SHADOW VIC II CHIP REGISTERS
1140 S1SPRX *=*+8             ;X LSB / SET1
1150 S2SPRX *=*+8             ;X LSB / SET2
1160 ;
1170 S1XMSB *=*+1             ;X MSB / SET1
1180 S2XMSB *=*+1             ;X MSB / SET2
1190 ;
1200 S1SPRY *=*+8             ;Y POS / SET1
1210 S2SPRY *=*+8             ;Y POS / SET2
1220 ;
1230 S1SPTR *=*+8             ;SPR POINTERS / SET1
1240 S2SPTR *=*+8             ;SPR POINTERS / SET2
1250 ;
1260 S1COLR *=*+8             ;SPR COLORS / SET1
1270 S2COLR *=*+8             ;SPR COLORS / SET2
1280 ;
1290 S1XPND *=*+1             ;X EXPAND / SET1
1300 S2XPND *=*+1             ;X EXPAND / SET2
1310 ;
1320 S1YPND *=*+1             ;Y EXPAND / SET1
1330 S2YPND *=*+1             ;Y EXPAND / SET2
1340 ;
1350 S1SCOL *=*+1             ;SPR-SPR COLLISIONS / SET1
1360 S2SCOL *=*+1             ;SPR-SPR COLLISIONS / SET2
1370 ;
1380 S1SDCL *=*+1             ;SPR-DATA COLLISIONS / SET1
1390 S2SDCL *=*+1             ;SPR-DATA COLLISIONS / SET2
1400 ;
1410 ;
1420 ;CONSTANTS
1430 SPOPTR =$07F8            ;SPR DATA POINTERS
1440 CIACRA =$DC0E            ;TIMER A CONTROL REGISTER
1450 ;
1460 ;VIC II CHIP REGISTERS
1470 SP0X   =$D000            ;SPR0 X POSITION LSB
1480 SP0Y   =$D001            ;SPR0 Y POSITION
1490 MSIGX  =$D010            ;SPR X POSITION MSB'S
1500 RASTER =$D012            ;RASTER LSB
1510 SPENA  =$D015            ;SPR ENABLE REGISTER
1520 YXPAND =$D017            ;SPR Y EXPANSION
```

```
1530 VICIRQ =$D019           ;VIC IRQ REGISTER
1540 IRQMSK =$D01A           ;VIC IRQ ENABLE ENABLE
1550 SPDACL =$D01B           ;SPR-DATA COLLISION REGISTER
1560 XXPAND =$D01D           ;SPR X EXPANSION
1570 SPRCOL =$D01E           ;SPR-SPR COLLISION REGISTER
1580 SP0COL =$D027           ;SPR0 COLOR
1590 CIACRA =$DC0E           ;TIMER A CONTROL REGISTER
1600 ;
1610 ;
1620 *=$C000                 ;EXAMPLE ASSEMBLE VALUE
1630 ;
1640 ;**********************
1650 ;*    IRQSVC  SETUP   *
1660 ;**********************
1670 ;
1680 SETUP  SEI              ;DISABLE IRQ'S WHILE SETTING UP
1690        LDA #$00         ;DISABLE 6526 INTERRUPTS
1700        STA CIACRA
1710        LDA #<IRQSVC     ;CHANGE IRQ VECTOR TO NEW
1720        STA $0314        ;  IRQ ROUTINE
1730        LDA #>IRQSVC
1740        STA $0314+1
1750        LDA #$FA         ;SET FIRST INTERRUPT FOR RASTER $FA
1760        STA RASTER
1770        LDA #$01         ;MASK OUT ALL BUT RASTER INTERRUPTS
1780        STA IRQMSK
1790        LDA #$FF         ;ENABLE ALL SPRITES
1800        STA SPENA
1810        CLI              ;RE-ENABLE IRQ'S
1820 ;
1830 SETUP0 JMP SETUP0       ;GO INTO AN ENDLESS LOOP
1840 ;
1850 ;
1860 ;
1870 ;*****************************
1880 ;* IRQSVC: INTERRUPT SERVICE *
1890 ;*    FOR MULTIPLE SPRITES    *
1900 ;*****************************
1910 ;
1920 IRQSVC
1930        CLD              ;MAKE SURE DECIMAL MODE IS NOT SET
1940        LDA VICIRQ       ;CLEAR CURRENT INTERRUPT...
1950        BPL EXIT         ;...BUT MAKE SURE THE VIC CAUSED IT
1960        STA VICIRQ
1970        LDX #$0F         ;ASSUME SET2 IS TO BE SET UP...
1980        LDY #$01         ;   (BY PREPARING INDEXES)
1990        LDA #$FA         ;...UNLESS RASTER >= $FA
2000        CMP RASTER       ;(IN WHICH CASE DO NOT BRANCH...
2010        BNE IRQ0         ;...AND SET UP SET1)
2020 ;
2030 ;*** SET UP SET1 HALF OF SCREEN ***
2040 RASTOP LDX #$07         ;PREPARE INDEXES TO SET1
2050        DEY
2060        LDA #$98         ;NEXT INTERRUPT AT RASTER = $98
2070 ;
2080 ;
2090 IRQ0   STA RASTER       ;LATCH FOR NEXT RASTER INTERRUPT
2100        STX HOLDX        ;STORE .X FOR LATER USE
```

```
;
         LDA S1XMSB,Y    ;TRANSFER SHADOW...
         STA MSIGX       ;...X POSITION MSB'S
         LDA S1YPND,Y
         STA YXPAND      ;...Y EXPANSION
         LDA S1XPND,Y
         STA XXPAND      ;...X EXPANSION
;
         LDA SPRCOL      ;STORE SPR-SPR COLLISION REGISTER
         STA S1SCOL,Y
         LDA SPDACL      ;AND SPR-DATA COLLISION REGISTER
         STA S1SDCL,Y
;
         LDY #$0E        ;TRANSFER SHADOW...
IRQ1     LDA S1SPRX,X
         STA SP0X,Y      ;...X POSITION LSB'S
         LDA S1SPRY,X
         STA SP0Y,Y      ;...Y POSITION
         DEX
         DEY
         DEY
         BPL IRQ1
;
         LDX HOLDX
         LDY #$07        ;TRANSFER SHADOW...
IRQ2     LDA S1COLR,X
         STA SP0COL,Y    ;...SPRITE COLORS
         LDA S1SPTR,X
         STA SP0PTR,Y    ;...SPRITE POINTERS
         DEX
         DEY
         BPL IRQ2
;
;
EXIT     PLA             ;RESTORE REGISTERS
         TAY
         PLA
         TAX
         PLA
         RTI
;
.END
```

2. Hex dump (as assembled at $C000):

```
.: 2000 78 A9 00 8D 0E DC A9 23 8D 14 03 A9 C0 8D 15 03
.: 2010 A9 FA 8D 12 D0 A9 01 8D 1A D0 A9 FF 8D 15 D0 58
.: 2020 4C 20 C0 D8 AD 19 D0 10 59 8D 19 D0 A2 0F A0 01
.: 2030 A9 FA CD 12 D0 D0 05 A2 07 88 A9 98 8D 12 D0 86
.: 2040 02 B9 13 00 8D 10 D0 B9 47 00 8D 17 D0 B9 45 00
.: 2050 8D 1D D0 AD 1E D0 99 49 00 AD 1B D0 99 4B 00 A0
.: 2060 0E B5 03 99 00 D0 B5 15 99 01 D0 CA 88 88 10 F1
.: 2070 A6 02 A0 07 B5 35 99 27 D0 B5 25 99 F8 07 CA 88
.: 2080 10 F2 68 A8 68 AA 68 40
```

3. Data statements (as assembled at $C000):

```
1000 data 120,169,0,141,14,220,169,35,141,20,3,169,192,141,21,3
1010 data 169,250,141,18,208,169,1,141,26,208,169,255,141,21,208,88
1020 data 76,32,192,216,173,25,208,16,89,141,25,208,162,15,160,1
1030 data 169,250,205,18,208,208,5,162,7,136,169,152,141,18,208,134
1040 data 2,185,19,0,141,16,208,185,71,0,141,23,208,185,69,0
1050 data 141,29,208,173,30,208,153,73,0,173,27,208,153,75,0,160
1060 data 14,181,3,153,0,208,181,21,153,1,208,202,136,136,16,241
1070 data 166,2,160,7,181,53,153,39,208,181,37,153,248,7,202,136
1080 data 16,242,104,168,104,170,104,64
```

C. Memory/Register Requirements: The set up routine (SETUP) requires 32 ($20) bytes of memory (excluding the endless loop on line 1830) and uses the accumulator. The interrupt service routine, IRQSVC, requires 101 ($65) bytes and uses the accumulator, and the x and y registers. Also, 75 ($6b) bytes of zero page must be reserved for variables.

D. Worst case execution time is 40 cycles (39.08 micro-seconds at1.02 MHz) for SETUP (excluding line 1830) and 530 cycles (517.81 micro-seconds)for IRQSVC.

E. User Notes:

1. During each interrupt the sprite-to-sprite and sprite-to-data collision registers are stored for use in a collision routine.

2. Whenever any or all of a sprite from one set is

positioned in the part of the screen reserved for the other, problems may develop. At best only flickering will occur, but at worst sprites may be split or color and sprite data may change in the middle. Sprites expanded in y have limited mobility because of a greater likelihood of crossing boundries.

3. IRQSVC does not push any of the registers onto the stack upon interrupt. If executed on a Commodore 64, this is done by the KERNAL (assuming lines 1710-1740 are left intact). The MAX Machine (or a Commodore 64 which emulates a MAX), however, does not have a KERNAL. Therefore, the contents of locations $FFFE and $FFFF should contain the address of IRQSVC (low byte, high byte order). Further, at the start of the IRQ routine should be the following code to push the registers onto the stack:

```
PHA
TXA
PHA
TYA
PHA
```

F. Example: The following example enables sixteen sprites and moves them around the screen in the x direction. Sample sprite data has been provided on the devlopment disk under the name 'numbers.bin'. The data should be loaded from VICMON (XVM4.8).

```
1000 .PAGE 'IRQSVC EXAMPLE'
1010 ;
1020 ;THIS EXAMPLE OF THE IRQSVC ROUTINE ENABLES
1030 ;16 SPRITES AND MOVES THEM AROUND THE SCREEN
1040 ;IN THE X DIRECTION. FOR SAMPLE SPRITE DATA,
1050 ;LOAD THE BINARY FILE CALLED 'NUMBERS.BIN'
1060 ;WITH VICMON
1070 ;
1080 *=$02                    ;OR ELSEWHERE ON ZERO PAGE
```



```
1090 ;VARIABLES
1100 HOLDX  *=*+1             ;TEMPORARY STORAGE FOR X REGISTER
1110 ;
1120 ; SHADOW VIC II CHIP REGISTERS
1130 S1SPRX *=*+8             ;X LSB / SET1
1140 S2SPRX *=*+8             ;X LSB / SET2
1150 ;
1160 S1XMSB *=*+1             ;X MSB / SET1
1170 S2XMSB *=*+1             ;X MSB / SET2
1180 ;
1190 S1SPRY *=*+8             ;Y POS / SET1
1200 S2SPRY *=*+8             ;Y POS / SET2
1210 ;
1220 S1SPTR *=*+8             ;SPR POINTERS / SET1
1230 S2SPTR *=*+8             ;SPR POINTERS / SET2
1240 ;
1250 S1COLR *=*+8             ;SPR COLORS / SET1
1260 S2COLR *=*+8             ;SPR COLORS / SET2
1270 ;
1280 S1XPND *=*+1             ;X EXPAND / SET1
1290 S2XPND *=*+1             ;X EXPAND / SET2
1300 ;
1310 S1YPND *=*+1             ;Y EXPAND / SET1
1320 S2YPND *=*+1             ;Y EXPAND / SET2
1330 ;
1340 S1SCOL *=*+1             ;SPR-SPR COLLISIONS / SET1
1350 S2SCOL *=*+1             ;SPR-SPR COLLISIONS / SET2
1360 ;
1370 S1SDCL *=*+1             ;SPR-DATA COLLISIONS / SET1
1380 S2SDCL *=*+1             ;SPR-DATA COLLISIONS / SET2
1390 ;
1400 ;
1410 ;CONSTANTS
1420 SP0PTR =$07F8            ;SPR DATA POINTERS
1430 CIACRA =$DC0E            ;TIMER A CONTROL REGISTER
1440 ;
1450 ;VIC II CHIP REGISTERS
1460 SP0X   =$D000            ;SPR0 X POSITION LSB
1470 SP0Y   =$D001            ;SPR0 Y POSITION
1480 MSIGX  =$D010            ;SPR X POSITION MSB'S
1490 RASTER =$D012            ;RASTER LSB
1500 SPENA  =$D015            ;SPR ENABLE REGISTER
1510 YXPAND =$D017            ;SPR Y EXPANSION
1520 VICIRQ =$D019            ;VIC IRQ REGISTER
1530 IRQMSK =$D01A            ;VIC IRQ ENABLE ENABLE
1540 SPDACL =$D01B            ;SPR-DATA COLLISION REGISTER
1550 XXPAND =$D01D            ;SPR X EXPANSION
1560 SPRCOL =$D01E            ;SPR-SPR COLLISION REGISTER
1570 SP0COL =$D027            ;SPR0 COLOR
1580 CIACRA =$DC0E            ;TIMER A CONTROL REGISTER
1590 ;
1600 ;
1610 *=$C000                  ;EXAMPLE ASSEMBLE VALUE
1620 ;
1630 ;**********************
1640 ;*     IRQSVC  SETUP  *
1650 ;**********************
1660 ;
```

```
EXAM    SEI              ;DISABLE IRQ'S WHILE SETTING UP
        LDA #$00         ;DISABLE 6526 INTERRUPTS
        STA CIACRA
        LDA #<IRQSVC     ;CHANGE IRQ VECTOR TO NEW
        STA $0314        ;  IRQ ROUTINE
        LDA #>IRQSVC
        STA $0314+1
        LDA #$FA         ;SET FIRST INTERRUPT FOR RASTER $FA
        STA RASTER
        LDA #$01         ;MASK OUT ALL BUT RASTER INTERRUPTS
        STA IRQMSK
        LDA #$FF         ;ENABLE ALL SPRITES
        STA SPENA
;
        LDY #69          ;LOAD TWO SETS OF SPRITE DATA INTO
EXAM0   LDA VICDAT,Y     ;  SHADOW VIC DATA VARIABLES
        STA S1SPRX,Y
        DEY
        BPL EXAM0
;
        CLI              ;RE-ENABLE INTERRUPTS
;
EXAM1   LDX #$0F         ;MOVE EACH SPR RIGHT 1 POSITION
EXAM2   INC S1SPRX,X
        DEX
        BPL EXAM2
;
        LDX #$0F
EXAM3   LDY #$FF         ;A DELAY LOOP
EXAM4   DEY
        BNE EXAM4
        DEX
        BNE EXAM3
;
        JMP EXAM1
;
;
;*****************************
;* IRQSVC: INTERRUPT SERVICE *
;*    FOR MULTIPLE SPRITES   *
;*****************************
;
IRQSVC
        CLD              ;MAKE SURE DECIMAL MODE IS NOT SET
        LDA VICIRQ       ;CLEAR CURRENT INTERRUPT...
        BPL EXIT         ;...BUT MAKE SURE THE VIC CAUSED IT
        STA VICIRQ
        LDX #$0F         ;ASSUME SET2 IS TO BE SET UP...
        LDY #$01         ;   (BY PREPARING INDEXES)
        LDA #$FA         ;...UNLESS RASTER >= $FA
        CMP RASTER       ;(IN WHICH CASE DO NOT BRANCH...
        BNE IRQ0         ;...AND SET UP SET1)
;
;*** SET UP SET1 HALF OF SCREEN ***
RASTOP  LDX #$07         ;PREPARE INDEXES TO SET1
        DEY
        LDA #$99         ;NEXT INTERRUPT AT RASTER = $98
;
```



```
;
IRQ0    STA RASTER      ;LATCH FOR NEXT RASTER INTERRUPT
        STX HOLDX       ;STORE .X FOR LATER USE
;
        LDA S1XMSB,Y    ;TRANSFER SHADOW...
        STA MSIGX       ;...X POSITION MSB'S
        LDA S1YPND,Y
        STA YXPAND      ;...Y EXPANSION
        LDA S1XPND,Y
        STA XXPAND      ;...X EXPANSION
;
        LDA SPRCOL      ;STORE SPR-SPR COLLISION REGISTER
        STA S1SCOL,Y
        LDA SPDACL      ;AND SPR-DATA COLLISION REGISTER
        STA S1SDCL,Y
;
        LDY #$0E        ;TRANSFER SHADOW...
IRQ1    LDA S1SPRX,X
        STA SP0X,Y      ;...X POSITION LSB'S
        LDA S1SPRY,X
        STA SP0Y,Y      ;...Y POSITION
        DEX
        DEY
        DEY
        BPL IRQ1
;
        LDX HOLDX
        LDY #$07        ;TRANSFER SHADOW....
IRQ2    LDA S1COLR,X
        STA SP0COL,Y    ;...SPRITE COLORS
        LDA S1SPTR,X
        STA SP0PTR,Y    ;...SPRITE POINTERS
        DEX
        DEY
        BPL IRQ2
;
;
EXIT    PLA             ;RESTORE REGISTERS
        TAY
        PLA
        TAX
        PLA
        RTI
;
;
;
VICDAT ;SAMPLE SETS OF SPRITE DATA
        .BYT $18,$30,$48,$60,$78,$90,$A8,$C0 ;X LSB'S / SET1
        .BYT $18,$30,$48,$60,$78,$90,$A8,$C0 ;X LSB'S / SET2
        .BYT $00,$00 ;X MSB'S / SET1 & SET2
        .BYT $32,$36,$3A,$3E,$42,$46,$4A,$4E ;Y POS'S / SET1
        .BYT $E5,$E1,$DD,$D9,$D5,$D1,$CD,$C9 ;Y POS'S / SET2
        .BYT $C0,$C1,$C2,$C3,$C4,$C5,$C6,$C7 ;DATA POINTERS / SET1
        .BYT $C8,$C9,$CA,$CB,$CC,$CD,$CE,$CF ;DATA POINTERS / SET2
        .BYT $00,$01,$02,$03,$04,$05,$07,$07 ;SPR COLORS / SET1
        .BYT $08,$09,$0A,$0B,$0C,$0D,$0E,$0F ;SPR COLORS / SET2
        .BYT $00,$00 ;X EXPANSION / SET1 & SET2
        .BYT $00,$00 ;Y EXPANSION / SET1 & SET2
```

```
2830 ;
2840 .END
```

SOFTWARE APPLICATION
NOTE 1005B

AUTHOR : Andy Finkel

SUBJECT : Multi-sprite processor

TV STANDARD : NTSC or PAL

ABSTRACT

This applications note describes a method for displaying up to 32 sprites on the screen at the same time through software interrupt control. Four 'shadow' VIC-II chips are created and maintained by the multi-sprite routine. The shadow registers on the chips are mapped into the actual VIC-II hardware sprite registers under interrupt control.

EXPOSITION

The multi-sprite routine is an IRQ driven interrupt routine which can maintain up to 4 shadow VIC-II chips. Each of the shadow chips is assigned an area of the screen by the programmer. Each chip has the use of the 8 sprites in its section of screen, as well as control of the background color. Any sprites used by a shadow chip must stay in the area controlled by that shadow chip. If this rule is violated, it is possible that the sprite will not be displayed. The following charts shows the area of control for each of the 4 shadow VIC-II chips.

CHIP	RASTER LINE OF INTERRUPT (CEN)	AREA OF CONTROL
0	48	49-100
1	100	101-160
2	160	161-199
3	199	200-48 (wraps around to the top of the screen)

Note that the bottom of the area of control of each chip is defined by the start of the area of control of the next chip. i.e. the bottom of chip 1's area is defined as the beginning of chip 2's area. The areas of control can be changed dynamically, but should be at least 21 raster lines apart when using unexpanded (in Y) sprites, or 42 raster lines apart when using Y expanded sprites.

There are four main control locations, one for each of the shadow chips, called CEN (Chip ENable) in the routine. If one of the CEN locations contains a 0, that chip is DISABLED. Otherwise, each is to be set to one before the desired raster line of the area of control for the chip. The background color is changed one raster line later. The new sprites are activated several raster lines after that.

LIMITS

Care must be taken that a sprite maintained by one pseudo VIC chip does not wander into the area under the control of another VIC chip. If this happens, one of the sprites will not be displayed.

Generally, the sprite that belongs in that area will have priority.

If it is desired to have a sprite that can go anywhere on the screen, the easiest method is to duplicate the parameters (position, color, and pointer) in each of the VIC chips. That way, the appropriate VIC chip will display the sprite.

SOURCE LISTING

```
1000 .PAG   'DPROC'
1010 ;
1020 ;***********************************
1030 ;* LIST PROCESSOR
1040 ;***********************************
1050 ;
1060 *=$0002
1070 CEN     *=*+4              ;CONTROL FLAGS FOR CHIPS
1080 ;
1090 CHIP    *=*+1
1100 ;
1110 ;SHADOW VIC CHIP REGISTERS
1120 ;
1130 PSPR0   *=*+16             ;LSB FOR CHIP #1
1140 PMSBX0  *=*+1              ;X MSB FOR CHIP #1
1150 PSPR1   *=*+16             ;LSB FOR CHIP #2
1160 PMSBX1  *=*+1              ;X MSB FOR CHIP #2
1170 PSPR2   *=*+16             ;LSB FOR CHIP #3
1180 PMSBX2  *=*+1              ;X MSB FOR CHIP #3
1190 PSPR3   *=*+16             ;LSB FOR CHIP #4
1200 PMSBX3  *=*+1              ;X MSB FOR CHIP #4
1210 ;
1220 PSPTR0  *=*+8              ;SPRITE POINTERS FOR CHIP #1
1230 PSPTR1  *=*+8              ;SPRITE POINTERS FOR CHIP #2
1240 PSPTR2  *=*+8              ;SPRITE POINTERS FOR CHIP #3
1250 PSPTR3  *=*+8              ;SPRITE POINTERS FOR CHIP #4
1260 ;
1270 PSCLR0  *=*+8              ;SPRITE COLORS FOR CHIP #1
1280 PSCLR1  *=*+8              ;SPRITE COLORS FOR CHIP #2
1290 PSCLR2  *=*+8              ;SPRITE COLORS FOR CHIP #3
1300 PSCLR3  *=*+8              ;SPRITE COLORS FOR CHIP #4
1310 ;
1320 COLOR0  *=*+1              ;COLOR FOR CHIP #0
1330 COLOR1  *=*+1              ;COLOR FOR CHIP #1
1340 COLOR2  *=*+1              ;COLOR FOR CHIP #2
1350 COLOR3  *=*+1              ;COLOR FOR CHIP #3
1360 ;
1370 PSCOL0  *=*+1              ;SPR-SPR COLLISIONS FOR CHIP #1
1380 PSCOL1  *=*+1              ;SPR-SPR COLLISIONS FOR CHIP #2
1390 PSCOL2  *=*+1              ;SPR-SPR COLLISIONS FOR CHIP #3
1400 PSCOL3  *=*+1              ;SPR-SPR COLLISIONS FOR CHIP #4
1410 ;
1420 PSDCL0  *=*+1              ;SPR-DATA COLLISIONS FOR CHIP #1
1430 PSDCL1  *=*+1              ;SPR-DATA COLLISIONS FOR CHIP #2
1440 PSDCL2  *=*+1              ;SPR-DATA COLLISIONS FOR CHIP #3
1450 PSDCL3  *=*+1              ;SPR-DATA COLLISIONS FOR CHIP #4
1460 ;
1470 ;* VIC CHIP REGISTERS
1480 ;
1490 SPRX    =$D000
1500 SPRY    =$D001
```

```
1510 MSIGX  =$D010
1520 RASTER =$D012
1530 SPENA  =$D015
1540 VIRQ   =$D019
1550 VENABL =$D01A
1560 SPRCOL =$D01E
1570 SPDACL =$D01F
1580 BCOLOR =$D021
1590 SPRCLR =$D027
1600 ;
1610 POINT  =2040              ;SPRITE POINTERS
1620 ;CONSTANTS
1630 ;
1640 IRVAL  =47                ;INTERRUPT 1 RASTER LINE BEFORE ...
1650 ;
1660 *=$C000
1670 ;
1680 INIT    SEI
1690         LDA $DC0E         ;TURN OFF TIMER
1700         AND #254
1710         STA $DC0E
1720         LDA $DC0D         ;CLEAR ANY LAST INTERRUPTS
1730 ;
1740         LDA #>DISPLY      ;SET NEW INTERRUPT VECTOR
1750         STA $0315
1760         LDA #<DISPLY
1770         STA $0314
1780 ;
1790         LDA #IRVAL        ;SET UP CHIP ENABLE FLAGS
1800         STA CEN           ;(CHIP 0 SHOULD ALWAYS BE IRVAL)
1810         LDA #90
1820         STA CEN+1
1830         LDA #130
1840         STA CEN+2
1850         LDA #170
1860         STA CEN+3
1870 ;
1880         LDA CEN           ;SET FOR 1ST INTERRUPT
1890         STA RASTER
1900         LDA #$00
1910         STA CHIP
1920 ;
1930         LDA #$1           ;ENABLE INTERRUPTS ON RASTER
1940         STA VENABL
1950 ;
1960         LDX #14           ;SET THE Y REGISTERS
1970 LAY     LDA #$40
1980         STA PSPR0+1,X
1990         LDA #100
2000         STA PSPR1+1,X
2010         LDA #160
2020         STA PSPR2+1,X
2030         LDA #$E0
2040         STA PSPR3+1,X
```

```
2050 ;
2060          TXA              ;DIVIDE BY 2
2070          LSR A
2080          TAY
2090 ;
2100          LDA XTAB,Y       ;SET THE X REGISTERS
2110          STA PSPR0,X
2120          STA PSPR1,X
2130          STA PSPR2,X
2140          STA PSPR3,X
2150 ;
2160          DEX
2170          DEX
2180          BPL LAY
2190 ;
2200          LDY #31
2210 PAG      LDA #192         ;SET THE SPRITE POINTERS
2220          STA PSPTR0,Y
2230 ;
2240          TYA              ;COLOR IT
2250          STA PSCLR0,Y
2260          DEY
2270          BPL PAG
2280 ;
2290          LDA #0
2300          STA PMSBX0
2310          STA PMSBX1
2320          STA PMSBX2
2330          STA PMSBX3
2340 ;
2350          LDY #3           ;SET THE COLOR REGS
2360 LAG      TYA
2370          STA COLOR0,Y
2380          DEY
2390          BPL LAG
2400 ;
2410          LDY #62          ;CREATE THE SPRITE BLOCK
2420          LDA #255
2430 CS1      STA $3000,Y
2440          DEY
2450          BPL CS1
2460 ;
2470          LDA #255         ;ACTIVATE ALL SPRITES
2480          STA SPENA
2490 ;
2500          CLI              ;START THINGS OFF
2510 ;
2520 HERE     LDX #14
2530 HAG      INC PSPR0,X
2540          INC PSPR1,X
2550          INC PSPR2,X
2560          INC PSPR3,X
2570 ;
2580          DEX
```

```
2590         DEX
2600         BPL HAG
2610 ;
2620         LDX #50          ;NOW WAIT
2630 W1      LDY #255
2640 W2      DEY
2650         BNE W2
2660         DEX
2670         BNE W1
2680         BEQ HERE         ;ALWAYS
2690 ;
2700 XTAB    .BYT 24,49,73,99,124,149,164,200
2710 ;
2720 ;***********************************
2730 ;* CHIP LIST PROCESSOR
2740 ;***********************************
2750 ;
2760 DISPLY  LDA VIRQ         ;ASSUME THE VIC CHIP CAUSED THE INTERRUPT
2770 ;
2780         STA VIRQ         ;CLEAR THE INTERRUPT REGISTER
2790 ;
2800         LDY CHIP         ;GET THE CHIP WE ARE ON
2810 ;
2820         LDA COLORO,Y     ;GET NEW COLOR
2830         LDX RASTER       ;WAIT BEFORE COLOR CHANGE
2840 WAIT    CPX RASTER
2850         BEQ WAIT
2860         STA BCOLOR       ;CHANGE BACKGROUND COLOR
2870 ;
2880 D1      INY              ;GET SET FOR NEXT LINE
2890 ;
2900         TYA              ;DO A MOD 3
2910         AND #$03         ;SINCE THERE ARE 3 CHIPS
2920         TAY
2930 ;
2940         LDA CEN,Y        ;GET THE RASTER FOR NEXT CHIP
2950         BEQ D1           ;THIS CHIP IS NOT IN USE
2960 ;
2970         STA RASTER       ;SET THE COMPARE REGISTER
2980 ;
2990         LDX CHIP         ;SELECT PSEUDO CHIP
3000         STY CHIP         ;SET FOR NEXT PSEUDO CHIP
3010 ;
3020         CPX #00          ;WHICH CHIP ARE WE DOING NOW ?
3030         BEQ VC0          ;CHIP 0
3040         CPX #$1
3050         BEQ VC1          ;CHIP 1
3060         CPX #$2
3070         BEQ VC2          ;CHIP 2
3080 ;
3090 VC3     LDA SPRCOL       ;SAVE OLD COLLISION REGISTERS
3100         STA PSCOL3
3110 ;
3120         LDA SPDACL
```

```
3130          STA PSDCL3
3140 ;
3150          LDX #16
3160 VC03     LDA PSPR3,X
3170          STA SPRX,X
3180          DEX
3190          BPL VC03
3200 ;
3210          LDX #7
3220 ;
3230 VC003    LDA PSPTR3,X     ;GET PSEUDO SPRITE POINTER
3240          STA POINT,X      ;SET REAL ONE
3250          LDA PSCLR3,X     ;GET PSEUDO SPRITE COLOR
3260          STA SPRCLR,X     ;SET REAL REGS
3270          DEX
3280          BPL VC003
3290 ;
3300          BMI DEXIT        ;ALWAYS
3310 ;
3320 VC2      LDA SPRCOL       ;SAVE OLD COLLISION REGISTERS
3330          STA PSCOL2
3340 ;
3350          LDA SPDACL
3360          STA PSDCL2
3370 ;
3380          LDX #16
3390 VC02     LDA PSPR2,X
3400          STA SPRX,X
3410          DEX
3420          BPL VC02
3430 ;
3440          LDX #7
3450 ;
3460 VC002    LDA PSPTR2,X     ;GET PSEUDO SPRITE POINTER
3470          STA POINT,X      ;SET REAL ONE
3480          LDA PSCLR2,X     ;GET PSEUDO SPRITE COLOR
3490          STA SPRCLR,X     ;SET REAL REGS
3500          DEX
3510          BPL VC002
3520 ;
3530          BMI DEXIT        ;ALWAYS
3540 ;
3550 VC1      LDA SPRCOL       ;SAVE OLD COLLISION REGISTERS
3560          STA PSCOL1
3570 ;
3580          LDA SPDACL
3590          STA PSDCL1
3600 ;
3610          LDX #16
3620 VC01     LDA PSPR1,X
3630          STA SPRX,X
3640          DEX
3650          BPL VC01
3660 ;
```

```
3670            LDX #7
3680 VC001      LDA PSPTR1,X      ;GET PSEUDO SPRITE POINTER
3690            STA POINT,X       ;SET REAL ONE
3700            LDA PSCLR1,X      ;GET PSEUDO SPRITE COLOR
3710            STA SPRCLR,X      ;SET REAL REGS
3720            DEX
3730            BPL VC001
3740 ;
3750            BMI DEXIT         ;ALWAYS
3760 ;
3770 VCO        LDA SPRCOL        ;SAVE OLD COLLISION REGISTERS
3780            STA PSCOLO
3790 ;
3800            LDA SPDACL
3810            STA PSDCLO
3820 ;
3830            LDX #16
3840 VCOO       LDA PSPRO,X
3850            STA SPRX,X
3860            DEX
3870            BPL VCOO
3880 ;
3890            LDX #7
3900 ;
3910 VCOOO      LDA PSPTRO,X      ;GET PSEUDO SPRITE POINTER
3920            STA POINT,X       ;SET REAL ONE
3930            LDA PSCLRO,X      ;GET PSEUDO SPRITE COLOR
3940            STA SPRCLR,X      ;SET REAL REGS
3950            DEX
3960            BPL VCOOO
3970 ;
3980 DEXIT      PLA               ;RESTORE REGISTERS
3990            TAY
4000            PLA
4010            TAX
4020            PLA
4030            RTI
4040 ;
4050 .END
```

HEX DUMP

```
.: C000    78 AD 0E DC 29 FE 8D 0E
.: C008    DC AD 0D DC A9 C0 8D 15
.: C010    03 A9 A8 8D 14 03 A9 2F
.: C018    85 02 A9 5A 85 03 A9 02
.: C020    85 04 A9 AA 85 05 A5 02
.: C028    8D 12 D0 A9 00 85 06 A9
.: C030    01 8D 1A D0 A2 0E A9 40
.: C038    95 08 A9 64 95 19 A9 A0
.: C040    95 2A A9 E0 95 38 8A 4A
```

```
.: C048   A8 B9 A3 C0 95 07 95 18
.: C050   95 29 95 3A CA CA 10 DE
.: C058   A0 1F A9 C0 99 4B 00 98
.: C060   99 6B 00 88 10 F4 A9 00
.: C068   85 17 85 28 85 39 85 4A
.: C070   A0 03 98 99 8B 00 88 10
.: C078   F9 A0 3E A9 FF 99 00 30
.: C080   88 10 FA A9 FF 8D 15 D0
.: C088   58 A2 0E F6 07 F6 18 F6
.: C090   29 F6 3A CA CA 10 F4 A2
.: C098   32 A0 FF 88 D0 FD CA D0
.: C0A0   F8 F0 E6 18 31 49 63 7C
.: C0A8   95 A4 C8 AD 19 D0 8D 19
.: C0B0   D0 A4 06 B9 88 00 AE 12
.: C0B8   D0 EC 12 D0 F0 FB 8D 21
.: C0C0   D0 C8 98 29 03 A8 B9 02
.: C0C8   00 F0 F6 8D 12 D0 A6 06
.: C0D0   84 06 E0 00 F0 77 E0 01
.: C0D8   F0 4E E0 02 F0 25 AD 1E
.: C0E0   D0 85 92 AD 1F D0 85 96
.: C0E8   A2 10 B5 3A 9D 00 D0 CA
.: C0F0   10 F8 A2 07 B5 63 9D F8
.: C0F8   07 B5 83 9D 27 D0 CA 10
.: C100   F3 30 6D AD 1E D0 85 92
.: C108   AD 1F D0 85 96 A2 10 B5
.: C110   29 9D 00 D0 CA 10 F8 A2
.: C118   07 B5 58 9D F8 07 B5 78
.: C120   9D 27 D0 CA 10 F3 30 48
.: C128   AD 1E D0 85 90 AD 1F D0
.: C130   85 94 A2 10 B5 18 9D 00
.: C138   D0 CA 10 F8 A2 07 B5 53
.: C140   9D F8 07 B5 73 9D 27 D0
.: C148   CA 10 F3 30 23 AD 1E D0
.: C150   85 8F AD 1F D0 85 93 A2
.: C158   10 B5 07 9D 00 D0 CA 10
.: C160   F8 A2 07 B5 4B 9D F8 07
.: C168   B5 6B 9D 27 D0 CA 10 F3
.: C170   68 A8 68 AA 68 40
```

DATA STATEMENTS

```
DATA   120, 173, 14, 220, 41, 254, 141, 14
DATA   220, 173, 13, 220, 169, 192, 141, 21
DATA   3, 169, 171, 141, 20, 3, 169, 47
DATA   133, 2, 169, 90, 133, 3, 169, 130
DATA   133, 4, 169, 170, 133, 5, 165, 2
DATA   141, 18, 208, 169, 0, 133, 6, 169
DATA   1, 141, 26, 208, 162, 14, 169, 64
DATA   149, 8, 169, 100, 149, 25, 169, 160
DATA   149, 42, 169, 224, 149, 59, 138, 74
DATA   168, 185, 163, 192, 149, 7, 149, 24
DATA   149, 41, 149, 58, 202, 202, 16, 222
DATA   160, 31, 169, 192, 153, 75, 0, 152
DATA   153, 107, 0, 136, 16, 244, 169, 0
```

```
DATA   133, 23, 133, 40, 133, 57, 133, 74
DATA   160, 3, 152, 153, 139, 0, 136, 16
DATA   249, 160, 62, 169, 255, 153, 0, 48
DATA   136, 16, 250, 169, 255, 141, 21, 208
DATA   88, 162, 14, 246, 7, 246, 24, 246
DATA   41, 246, 58, 202, 202, 16, 244, 162
DATA   50, 160, 255, 136, 208, 253, 202, 208
DATA   248, 240, 230, 24, 49, 73, 99, 124
DATA   149, 164, 200, 173, 25, 208, 141, 25
DATA   208, 164, 6, 185, 139, 0, 174, 18
DATA   208, 236, 18, 208, 240, 251, 141, 33
DATA   208, 200, 152, 41, 3, 168, 185, 2
DATA   0, 240, 246, 141, 18, 208, 166, 6
DATA   132, 6, 224, 0, 240, 119, 224, 1
DATA   240, 78, 224, 2, 240, 37, 173, 30
DATA   208, 133, 146, 173, 31, 208, 133, 150
DATA   162, 16, 181, 58, 157, 0, 208, 202
DATA   16, 248, 162, 7, 181, 99, 157, 248
DATA   7, 181, 131, 157, 39, 208, 202, 16
DATA   243, 48, 109, 173, 30, 208, 133, 146
DATA   173, 31, 208, 133, 150, 162, 16, 181
DATA   41, 157, 0, 208, 202, 16, 248, 162
DATA   7, 181, 91, 157, 248, 7, 181, 123
DATA   157, 39, 208, 202, 16, 243, 48, 72
DATA   173, 30, 208, 133, 144, 173, 31, 208
DATA   133, 148, 162, 16, 181, 24, 157, 0
DATA   208, 202, 16, 248, 162, 7, 181, 83
DATA   157, 248, 7, 181, 115, 157, 39, 208
DATA   202, 16, 243, 48, 35, 173, 30, 208
DATA   133, 143, 173, 31, 208, 133, 147, 162
DATA   16, 181, 7, 157, 0, 208, 202, 16
DATA   248, 162, 7, 181, 75, 157, 248, 7
DATA   181, 107, 157, 39, 208, 202, 16, 243
DATA   104, 168, 104, 170, 104, 64
```

SOFTWARE APPLICATION
NOTE 2001

Authors: Joe McEnerney and Eric Cotton

Subject: Addressing the Video and Color Matrices

Television Standard: NTSC or PAL

I. Abstract: The purpose of this application note is to supply a method of addressing the video and color matrices by specifying row and column numbers for the matrix element of concern. This will be achieved using the indirect indexed addressing mode of the 650x microprocessor and an auxiliary table.

II. EXPOSITION: In order to address an arbitrary element of any matrix of bytes stored in memory the user must have a base address B, a row number R, a column number C, the number of rows NR, and the number of columns NC. The number R must satisfy the inequality 0 <= R <= NR-1 and C must satisfy 0 <= C <= NC-1. The address, AE, of an arbitrary element can be found by performing the calculation

AE = R*NC+C+B.

This formula requires one multiply and two additions to find A. This is a rather high price to pay in execution time and memory. To solve these problems one can store the precalculated values R*NC+B, 0 <= R <= NR=1 in a table and use C as an index value. These table entries are, in fact, the addresses of the first element of each row of the matrix. Now, in our application of this principle, the matrix has 25 rows (NR=25) and 40 columns (NC=40). The base address is chosen by the user by selecting the high address bits VM13, ..., VM10 in register 24 ($18) of the 6566/6567 VIC II chip. By pre-calculating the table of addresses

40*R, 0 <= R <= 24

and using C as the value in the y index register, an arbitrary location in the video matrix can be accessed. In what follows, the value of B will be calculated from from VIC register 24. Moreover, the user must also note that the matrix of color nybbles has $D800 as a base address.

A. Program listings:

1. Source:

```
1000 .PAGE 'SET POINTERS'
1010 ;
1020 ;  SET UP POINTERS TO VIDEO & COLOR MATRICES
1030 ;            USING [VPTR] & [CPTR]
1040 ;
1050 ;SET VIDEO & COLOR MATRIX POINTERS USING THE
1060 ;VALUE IN THE X REGISTER AS A ROW NUMBER
1070 ;
1080           *=$0020          ;OR SOMEWHERE IN PG 0
1090 VPTR      *=*+2            ;POINTS AT VIDEO MATRIX
1100 CPTR      *=*+2            ;POINTS AT COLOR MATRIX
1110 VIC       =$D000           ;START OF VIC REGISTERS
1120 ;
1130 ;         *=$3000          ;FOR TEST ASSEMBLY
1140 ;
1150 SETPTR LDA RALT,X          ;GET THE ROW ADDRESS LOW
1160           STA VPTR         ;STORE IN VIDEO POINTER LOW
1170           STA CPTR         ;STORE IN COLOR POINTER LOW
1180           LDA VIC+24       ;GET THE VM BASE ADDRESS
1190           AND #$F0         ;CLEAR LOW BITS
1200           LSR A            ;RIGHT SHIFT IT TWICE
1210           LSR A            ;TO ALIGN IT PROPERLY
1220           ORA RAHT,X       ;OR IN THE HIGH OFFSET
1230           STA VPTR+1       ;STORE IN VIDEO POINTER HIGH
1240           AND #3           ;CLEAR ALL BUT 2 LSBS
1250           ORA #$D8         ;OR IN COLOR BASE HIGH
1260           STA CPTR+1       ;STORE IN COLOR POINTER HIGH
1270           RTS
1280 ;
1290 ;THESE ADDRESSES CORRESPOND TO SETTING THE VIDEO MATRIX
1300 ;STARTING ADDRESS TO $0000. THE FIRST ARRAY CONSISTS OF
1310 ;THE LOW ORDER BYTE OF THE ADDRESS OF THE FIRST BYTE OF
1320 ;EACH ROW OF THE VIDEO MATRIX. THE SECOND ARRAY CONSISTS
1330 ;OF THE HIGH ORDER ADDRESS BYTE. THESE VALUES ARE USED
1340 ;TO SET UP THE HIGH ORDER BYTE OF THE ADDRESS VIA ORING
1350 ;OF THE ALIGNED BASE WITH THE APPROPRIATE TABLE ENTRY
1360 ;
1370 RALT      .BYTE $00,$28,$50,$78,$A0,$C8,$F0,$18
1380           .BYTE $40,$68,$90,$B8,$E0,$08,$30,$58
1390           .BYTE $80,$A8,$D0,$F8,$20,$48,$70,$98
1400           .BYTE $C0
1410 RAHT      .BYTE 0,0,0,0,0,0,0,1,1,1,1,1,1,2,2,2
1420           .BYTE 2,2,2,2,3,3,3,3,3
1430 .END
```

2. Hex dump (as assembled at $3000):

```
.: 3000 BD 1A 30 85 20 85 22 AD 18 D0 29 F0 4A 4A 1D 33
.: 3010 30 85 21 29 03 09 D8 85 23 60 00 28 50 78 A0 C8
.: 3020 F0 18 40 68 90 B8 E0 08 30 58 80 A8 D0 F8 20 48
.: 3030 70 98 C0 00 00 00 00 00 00 00 01 01 01 01 01 01
.: 3040 02 02 02 02 02 02 02 03 03 03 03 03
```

3. Data statements (as assembled at $3000):

```
1000 data 189,26,48,133,32,133,34,173,24,208,41,240,74,74,29,51
1010 data 48,133,33,41,3,9,216,133,35,96,0,40,80,120,160,200
1020 data 240,24,64,104,144,184,224,8,48,88,128,168,208,248,32,72
1030 data 112,152,192,0,0,0,0,0,0,0,1,1,1,1,1,1
1040 data 2,2,2,2,2,2,2,3,3,3,3,3
```

C. Memory/Register requirements: This routine requires 76 ($4C) bytes of code including tables. It also expects the row number to be in the x register at JSR time. The accumulator is used but the y register is not.

D. Worst case execution time is 41 ($29) cycles. (At 1.02 mHz this is 40.2 micro-seconds.)

E. Examples:

1. To change a video matrix entry at ROW 10 and COL 15 to character number127 and the corresponding color code to 7 (yellow) one can use the following code:

```
LDX #10           ;(ROW #)
LDY #15           ;(COL #)
JSR SETPTR        ;SET UP POINTERS
LDA #127          ;CHR# 127
STA (VPTR),Y      ;STORE 127 @ VM(ROW, COL)
LDA #7            ;
STA (CPTR),Y      ;STORE 7 @ CM(ROW, COL)
```

2. To read the character code at ROW 10 and COL 15, one could use:

```
LDX #10           ;(ROW #)
LDY #15           ;(COL #)
JSR SETPTR        ;SET UP POINTERS
LDA (VPTR),Y      ;LOAD .A WITH VM(ROW, COL)
```

Software Application Note 2002

Authors: Jeff Bruette and Andy Finkel

Subject: Horizontal Scrolling

Television Standard: NTSC

I. Abstract: This scrolling routine performs all of the functions necessary to smoothly scroll the screen left or right. In one pass, it can scroll any increment from one bit to one byte. The main use of this routine would be for effects in which the landscape move by. Characters and color memory both move.

II. Exposition: The users program should call the routine labeled "SETIRQ". This will set up the routine variables and the raster interupt routine. Since the speed of the scroll is user definable, the user must store the speed in the variable called "SPEED". This may be done as often as necessary, but if the speed desired is to be constant, once is enough.

SPEED is expressed in two's compliment form and the valid ranges are 0 <= SPEED <= 8 or -8($f8) <= SPEED <= -1($ff). Therefore, moving $F8 scrolls 8 bits at a time from right to left while $08 scrolls 8 bits at a time from left to right. These numbers reflect the number of bits moved every 1/60th of a second (one screen scan of the raster). This holds true only if the routine labeled "BITMOV" is call at least once every 1/60th of a

second.

Assume that you character rows 3 ($03) through 12 ($0C) were to be scrolled, while all other rows were stationary. There is a formula to figure row/rasterline values. The formula is: CHARACTER ROW * 8 + $2A = TOP RASTER OF THAT ROW. In this example, the raster line for the top would be $42. This number should be placed in the constant that is named "TOP". The same formula should be used for calculating the bottom raster line but, add $07 to the answer. In this example, that number would be 12 ($0c). The answer is be $8A. Since we are scrolling rows 3 ($03) through 12 ($0C), there are 10 ($0A) lines being scrolled. Put this number in "ROWS".

The starting address of the top scroll row is also needed. This number minus 40 ($28) will give a two byte address. The high byte should be put in "HIBYTS". In this example, this would be an $04. The low byte is a $28. This should be put in "LOBYT". The only constant remaining is "HIBYTC". This is the high byte of the top of the color ram area being scrolled. Assuming that the start of screen memory is at 1024 ($0400), "HIBYTC" would be "HYBYTS" + $04. Put a $08 in the "HIBYTC" constant. Once these changes are made, this routine should be ready to use.

A note that should be made is that this routine must be used in the 38 column mode. This is an important point to make if the application note which deals with on screen detection is being used. Also, there are two changes to make in RASTWT if you are using multi-color mode. These changes are commented.

Another point which needs mentioning is a description of the YERPLT routine. This is a routine which must be supplied by the user. What it must do is place characters and/or colors which are

being introduced as new pictures. These character/colors should be put on the screen in the proper column. The proper column is automaticaly stored in the variable named COLUMN. An example of YERPLT is used in the demonstration program. This routine does not put up new information on the screen, but rather "wraps" information from one side of the screen around to the opposite side of the screen (i.e. puts column 0 into column 39 or puts column 39 into column 0).

Character Row / Memory Location Chart

Row No.	Mem. Loc.	Mem. Loc. - $28
$01	$0400	$0398
02	0428	0400
03	0450	0428
04	0478	0450
05	04A0	0478
06	04C8	04A0
07	04F0	04C8
08	0518	04F0
09	0540	0518
0A	0568	0540
0B	0590	0568
0C	05B8	0590
0D	05E0	05B8
0E	0608	05E0
0F	0630	0608
10	0658	0630
11	0680	0658
12	06A8	0680
13	06D0	06A8
14	06F8	06D0
15	0720	06F8
16	0748	0720
17	0770	0748
18	0798	0770
19	07C0	0798

Source Listing:

```
2640 ;***********************
2650 ;* SMOOTH SCROLL ROUTINE
2660 ;* WORKS FOR EITHER
2670 ;* RIGHT OR LEFT
2680 ;***********************
2690 ;
2700 ;* DECLARE THESE VARIABLES *
2710 *=$0002
2720 SPEED  *=*+1          ;SCROLLING SPEED
2730 SCRNLO *=*+2          ;LOW SCREEN ADDRESS
2740 SCRNHI *=*+2          ;HIGH SCREEN ADDRESS (LOW ADDRESS + 1)
2750 COLRLO *=*+2          ;LOW COLOR MEMORY ADDRESS
2760 COLRHI *=*+2          ;HIGH COLOR MEMORY ADDRESS (LOW ADD. + 1)
2770 DELAY1 *=*+1          ;"OPPERATION COMPLETE" FLAG
2780 XPMOV  *=*+1          ;SCROLX LOOKALIKE
2790 FLAG1  *=*+1
2800 COLUMN *=*+1          ;COLUMN WHICH NEEDS TO BE MODIFIED
2805 ROWS   =$25           ;NUMBER OF ROWS BEING SCROLLED
2810 HIBYTS =$03           ;HIGH BYTE OF SCREEN ADDRESS MINUS 40
2820 HIBYTC =$07           ;HIGH BYTE OF COLOR ADDRESS MINUS 40
2830 LOBYT  =$98           ;LOW BYTE OF SCREEN ADDRESS MINUS 40
```

```
2840 TOP     =$52                    ;VAL. OF RASTER LINE AT TOP OF SCROLL
2850 XEOR    =$F8                    ;TOP EORed WITH XEOR = BOTTOM RASTER
2851 IRQVEC =$314                    ;ON C64...CHANGE FOR GAME
2852 ;CIAICR =$DC0D                  ;THESE ARE IN STANDARD MAX DECLARE FILE
2853 ;RASTER =$D012
2854 ;VICIRQ =$D019
2855 ;SCROLX =$D016
2856 ;IRQMSK =$D01A
2857 ;CIACRA =$DC0E
2858 ;CIAPRA =$DC00
2860 *=$C000                         ;ASSEMBLE AT $C000
2870 SETIRQ  SEI                     ;TURN OFF ALL OUTSIDE INTERUPTS
2880         LDA  #$7F
2890         STA  CIAICR
2900         LDA  CIAICR
2910         LDA  #$01               ;SELECT RASTER TYPE INTERUPTS
2920         STA  IRQMSK
2930         LDA  #$00               ;ZERO OUT VARIABLES
2931         STA  XPMOVE             ;SCROLX LOOKALIKE
2940         STA  CIACRA
2950         STA  DELAY1
2960         STA  COLUMN
2970         LDA  #$TOP              ;RASTER LINE OF TOP OF SCROLL
2980         STA  RASTER
2990         STA  FLAG1
3000         LDA  #<RASTWT           ;GET THE LSB OF INTERUPT ROUTINE
3010         STA  IRQVEC
3020         LDA  #>RASTWT           ;GET THE MSB OF INTERUPT ROUTINE
3030         STA  IRQVEC+1
3040         LDA  #$C0               ;INITIALIZE THE 'SCROLX'
3050         STA  XPMOV              ;"LOOKALIKE" VARIABLE
3060         CLI
3070         RTS
3080 ;****** UPDATES ZERO PAGE SCROLX LOOKALIKE ******
3090 BITMOV  LDA  DELAY1             ;GET DELAY1 FLAG
3100         BEQ  BITRTS             ;BRANCH IF RASTER HASN'T MADE A PASS
3110         LDA  #$00               ;CLEAR OUT DELAY1 FLAG
3120         STA  DELAY1
3130         LDA  XPMOV              ;GET THE SCROLX LOOKALIKE
3140         CLC
3150         ADC  SPEED              ;ADD THE DISTANCE TO SCROLL. MUST BE:
3170         STA  XPMOV              ;STORE NEW OFFSET
3180         TAX
3190         AND  #$08               ;CHECK TO SEE IF IT WRAPPED
3200         BEQ  BITRTS             ;BRANCH IF IT HASN'T
3210         TXA
3220         AND  #$07               ;GET THE NEEDED BITS
3230         STA  XPMOV              ;STORE NEW NUMBER
3240         LDA  #$HIBYTS           ;HIGH BYTE OF STARTING...
3250         STA  SCRNLO+1           ;LINE MINUS 40
3260         STA  SCRNHI+1
3270         LDA  #$HIBYTC           ;HIGH BYTE OF COLOR RAM
3280         STA  COLRLO+1
3290         STA  COLRHI+1
3300         LDA  #$LOBYT            ;LOW BYTE OF STARTING...
3310         STA  SCRNLO             ;LINE MINUS 40
```

```
3320          TXA
3330          BMI  RLMOVE     ;WRAPPED PASSED ZERO, BRANCH TO RLMOVE
3340          BPL  LRMOVE     ;DIDN'T WRAP, XPMOV WAS JUST TOO BIG
3350 BITRTS   RTS
3360 ;****** MOVE SCREEN RIGHT TO LEFT ONE BYTE ******
3370 RLMOVE   LDX  #ROWS      ;NUMBER OF ROWS
3380 RLLP1    LDY  #$FF       ;COLUMN 0 MINUS 1
3390          LDA  SCRNLO
3400          CLC             ;GOTO NEXT ROW
3410          ADC  #40
3420          STA  SCRNLO
3430          STA  SCRNHI
3440          STA  COLRLO
3450          STA  COLRHI
3460          BCC  RLLP2      ;NO ADJUSTMENT WAS NEEDED, SO BRANCH
3470          INC  SCRNLO+1
3480          INC  SCRNHI+1
3490          INC  COLRLO+1
3500          INC  COLRHI+1
3510 RLLP2    INC  SCRNHI     ;ADJUST FOR THE ONE BYTE OFFSET IN MOVE
3520          INC  COLRHI
3530 RLLP3    INY             ;GOTO NEXT CHARACTER
3540          LDA  (SCRNHI),Y;GET A CHARACTER
3550          STA  (SCRNLO),Y;STORE CHARACTER IN NEW LOCATION
3560          LDA  (COLRHI),Y;GET A COLOR
3570          STA  (COLRLO),Y;STORE COLOR IN NEW LOCATION
3580          CPY  #39        ;IS ROW DONE?
3590          BNE  RLLP3      ;NO, SO BRANCH
3600          DEX
3610          BNE  RLLP1
3620          STY  COLUMN
3630          BEQ  YERPLT     ;BRANCH (or JMP) TO USER'S DRAW ROUTINE
3640 ;****** MOVE SCREEN LEFT TO RIGHT ONE BYTE ******
3650 LRMOVE   LDX  #ROWS      ;NUMBER OF ROWS
3660 LRLP1    LDY  #38        ;COLUMN 38
3670          LDA  SCRNLO
3680          CLC             ;GOTO NEXT ROW
3690          ADC  #40
3700          STA  SCRNLO
3710          STA  SCRNHI
3720          STA  COLRLO
3730          STA  COLRHI
3740          BCC  LRLP2      ;NO ADJUST NEEDED, SO BRANCH
3750          INC  SCRNLO+1
3760          INC  SCRNHI+1
3770          INC  COLRLO+1
3780          INC  COLRHI+1
3790 LRLP2    INC  SCRNHI     ;ADJUST FOR THE ONE BYTE OFFSET IN MOVE
3800          INC  COLRHI
3810 LRLP3    LDA  (SCRNLO),Y;GET A CHARACTER
3820          STA  (SCRNHI),Y;PUT CHARACTER IN NEW LOCATION
3830          LDA  (COLRLO),Y;GET A COLOR
3840          STA  (COLRHI),Y;STORE COLOR IN NEW LOCATION
3850          DEY             ;NEXT ONE?
3860          BPL  LRLP3      ;NO, SO BRANCH
3870          DEX
```

```
3880          BNE  LRLP1      ;GOTO NEXT ROW IF NOT DONE
3890          STX  COLUMN
3900          BEQ  YERPLT     ;BRANCH (or JMP) TO USER'S PLOT ROUTINE
3910 ;****** RASTER DRIVEN INTERUPT ROUTINE ******
3920 RASTWT   LDA  VICIRQ     ;CLEAR THE IRQ
3930          STA  VICIRQ
3940          AND  #$01       ;IS IT A RASTER INTERUPT
3950          BEQ  RASLP1     ;NO, SO BRANCH
3960          LDA  FLAG1      ;GET THE RASTER TOGGLE FLAG
3970          EOR  #XEOR      ;CALC. NEW RASTER LINE
4000          STA  FLAG1      ;STORE NEW STATUS
4010          STA  RASTER     ;SET THE NEXT RASTER LINE INTERUPT
4030          CMP  #TOP       ;IS RASTER AT TOP?
4040          BNE  RASLP1     ;NO, SO BRANCH
4050          LDA  XPMOV      ;GET THE SCROLX LOOKALIKE
4060          AND  #$07       ;GET JUST THE NEEDED BITS
4070          ORA  #$C0       ;OR #$D0 FOR MULTI-COLOR MODE
4080          STA  SCROLX     ;STORE IT IN THE VIC'S X SCROLL LOC.
4090          STA  DELAY1     ;SET DELAY1 FLAG <> 0
4100          BMI  RASLP2     ;ALWAYS
4110 RASLP1   LDA  #$C0       ;OR #$D0 FOR MCM
4120          STA  SCROLX
4130 RASLP2   PLA             ;RESET THE STACK
4140          TAY
4150          PLA
4160          TAX
4170          PLA
4180          RTI
4190 .END
```

Hex Dump:

```
.: C000   78 A9 7F 8D 0D DC AD 0D
.: C008   DC A9 01 8D 1A D0 A9 00
.: C010   8D 0E DC 85 0B 85 0E A9
.: C018   FB 8D 12 D0 85 0D A9 C3
.: C020   8D 14 03 A9 C0 8D 15 03
.: C028   A9 C0 85 0C 58 60 A5 0B
.: C030   F0 2A A9 00 85 0B A5 0C
.: C038   18 65 02 85 0C AA 29 08
.: C040   F0 1A 8A 29 07 85 0C A9
.: C048   03 85 04 85 06 A9 D7 85
.: C050   08 85 0A A9 D8 85 03 8A
.: C058   30 03 10 35 60 A2 19 A0
.: C060   FF A5 03 18 69 28 85 03
.: C068   85 05 85 07 85 09 90 08
.: C070   E6 04 E6 06 E6 08 E6 0A
.: C078   E6 05 E6 09 C8 B1 05 91
.: C080   03 B1 09 91 07 C0 27 D0
.: C088   F3 CA D0 D3 84 0E 4C 00
.: C090   00 A2 18 A0 26 A5 03 18
.: C098   69 28 85 03 85 05 85 07
.: C0A0   85 09 90 08 E6 04 E6 06
.: C0A8   E6 08 E6 0A E6 05 E6 09
.: C0B0   B1 03 91 05 B1 07 91 09
.: C0B8   88 10 F5 CA D0 D5 86 0E
.: C0C0   4C 00 00 AD 19 D0 8D 19
.: C0C8   D0 29 01 F0 1A A5 0D 49
.: C0D0   9B 85 0D 8D 12 D0 C9 FB
.: C0D8   D0 00 A5 0C 29 07 09 C0
.: C0E0   8D 16 D0 85 0B 30 05 A9
.: C0E8   C0 8D 16 D0 68 A8 68 AA
.: C0F0   68 40
```

Data Statements:

```
DATA    120,169,127,141,13,220,173,13
DATA    220,169,1,141,26,208,169,0
DATA    141,14,220,133,11,133,14,169
DATA    251,141,18,208,133,13,169,195
DATA    141,20,3,169,192,141,21,3
DATA    169,192,133,12,88,96,165,11
DATA    240,42,169,0,133,11,165,12
DATA    24,101,2,133,12,170,41,8
DATA    240,26,138,41,7,133,12,169
DATA    3,133,4,133,6,169,215,133
DATA    8,133,10,169,216,133,3,138
DATA    48,3,16,53,96,162,25,160
DATA    255,165,3,24,105,40,133,3
DATA    133,5,133,7,133,9,144,8
DATA    230,4,230,6,230,8,230,10
DATA    230,5,230,9,200,177,5,145
DATA    3,177,9,145,7,192,39,208
DATA    243,202,208,211,132,14,76,0
DATA    0,162,24,160,38,165,3,24
DATA    105,40,133,3,133,5,133,7
DATA    133,9,144,8,230,4,230,6
DATA    230,8,230,10,230,5,230,9
DATA    177,3,145,5,177,7,145,7
DATA    136,16,245,202,208,213,134,14
DATA    76,0,0,173,25,208,141,25
DATA    208,41,1,240,26,165,13,73
DATA    155,133,13,141,18,208,201,251
DATA    208,13,165,12,41,7,9,192
DATA    141,22,208,133,11,48,5,169
DATA    192,141,22,208,104,168,104,170
DATA    104,64
```

Execution Times:

Since the execution times vary depending on what area of the screen is being scrolled and the speed of the scroll, I have broken the routine into sections.

One pass only	SETIRQ	Always: 78 cycles
Once per frame	BITMOV	Best Case: 34 cycles Worst Case: 61 cycles (Worst case means that RLMOVE or LRMOVE is to be called because character boundary was crossed.)
Every 8th bit move	RLMOVE	Best Case: 1,118 cycles Worst Case: 28,733 cycles
Every 8th bit move	LRMOVE	Best Case: 1,118 cycles Worst Case: 28,733 cycles
Once every 1/60th. sec.	RASTWT	Always: 126 cycles

EXAMPLE

Here is a sample of smooth scrolling. This program will leave text on the top and bottom of the screen stationary. Using a joystick, control the speed and direction of the text in the center.

```
1000 ;
1010 ;***********************
1020 ;* EXAMPLE OF THE
1030 ;* SMOOTH SCROLL ROUTINE
1040 ;*   USES JOYSTICK
1050 ;***********************
1060 ;
1070 ;* DECLARE THESE VARIABLES *
1080 *=$0002  ;or somewhere in page zero
1090 SPEED  *=*+1
```

```
1100 SCRNLO *=*+2
1110 SCRNHI *=*+2
1120 COLRLO *=*+2
1130 COLRHI *=*+2
1140 DELAY1 *=*+1
1150 XPMOV  *=*+1
1160 FLAG1  *=*+1
1170 COLUMN *=*+1
1180 COUNT  *=*+1
1181 ;******* THESE ARE STANDARD MAX DECLARE VARIABLES *******
1190 CIAICR =$DC0D
1200 RASTER =$D012
1210 VICIRQ =$D019
1220 SCROLX =$D016
1230 IRQMSK =$D01A
1240 CIACRA =$DC0E
1250 CIAPRA =$DC00
1251 ;******* THESE ARE CONSTANTS ******
1252 ROWS =$09
1253 HIBYTS =$05
1254 HIBYTC =$d9
1255 LOBYT =$68
1256 XEOR =$40
1257 TOP =$82
1260 *=$2000
1270 SAMPLE  LDX  #0          ;CLEAR SCREEN/COLOR MEMORY
1280 POCS    LDA  #32         ;SEE APP. NOT 4007 (POWER ON CLEAR)
1290         STA  $0400,X
1300         STA  $0400+256,X
1310         STA  $0400+512,X
1320         STA  $0400+768,X
1330         LDA  #01
1340         STA  $D800,X
1350         STA  $D800+256,X
1360         STA  $D800+512,X
1370         STA  $D800+768,X
1380         DEX
1390         BNE  POC5
1400         LDY  #16         ;PUT UP MESSAGES
1410 NXTLIN  LDA  LINE1,Y
1420         STA  $0470+10,Y
1430         LDA  LINE2,Y
1440         STA  $04C0+10,Y
1450         LDA  LINE3,Y
1460         STA  $0720+10,Y
1470         LDA  LINE4,Y
1480         STA  $0770+10,Y
1490         LDA  LINE5,Y
1500         STA  $05B8+10,Y
1510         LDA  LINE6,Y
1520         STA  $06A8+10,Y
1530         DEY
1540         BPL  NXTLIN
1550         JSR  SETIRQ
1560         LDA  #$00        ;SET INITIAL SPEED
1570         STA  SPEED
```

```
1580 MAIN    JSR  BITMOV
1590         INC  COUNT
1600         LDX  #$00       ;GET JOYSTICK ***SEE APP. NOTE 4003 ***
1610         BNE  MAIN
1620         LDA  CIAPRA
1630         LSR  A
1640         LSR  A
1650         LSR  A
1660         BCS  JOYLP1
1670         DEX
1680 JOYLP1  LSR  A
1690         BCS  JOYEND
1700         INX
1710 JOYEND  TXA
1720         CLC
1730         ADC  SPEED
1740         CMP  #$F7       ;IF SPEED IS WITHIN RANGE, SAVE IT
1750         BEQ  MAIN
1760         CMP  #$09
1770         BEQ  MAIN
1780         STA  SPEED
1790         JMP  MAIN
1800 SETIRQ  SEI
1810         LDA  #$7F
1820         STA  CIAICR
1830         LDA  CIAICR
1840         LDA  #$01       ;SELECT RASTER TYPE INTERUPTS
1850         STA  IRQMSK
1860         LDA  #$00       ;ZERO OUT VARIABLES
1870         STA  $DC0E
1880         STA  DELAY1
1890         STA  COLUMN
1900         LDA  #$82       ;RASTER LINE OF TOP OF SCROLL
1910         STA  RASTER
1920         STA  FLAG1
1930         LDA  #<RASTWT   ;GET THE LSB OF INTERUPT ROUTINE
1940         STA  $314
1950         LDA  #>RASTWT   ;GET THE MSB OF INTERUPT ROUTINE
1960         STA  $315
1970         LDA  #$C0       ;INITIALIZE THE SCROLX LOOKALIKE VAR.
1980         STA  XPMOV
1990         CLI
2000         RTS
2010 ;***********************************************************************
2020 BITMOV  LDA  DELAY1     ;GET DELAY1 FLAG
2030         BEQ  BITRTS     ;BRANCH IF RASTER HASN'T MADE A PASS
2040         LDA  #$00       ;CLEAR OUT DELAY1 FLAG
2050         STA  DELAY1
2060         LDA  XPMOV      ;GET THE SCROLX LOOKALIKE
2070         CLC
2080         ADC  SPEED      ;ADD THE DISTANCE TO SCROLL. MUST BE:
2090                         ;$08 > X >= $00  OR  $FF >= X > $F8
2100         STA  XPMOV      ;STORE NEW OFFSET
2110         TAX
2120         AND  #$08       ;CHECK TO SEE IF IT WRAPPED
2130         BEQ  BITRTS     ;BRANCH IF IT HASN'T
```

```
2140            TXA
2150            AND   #$07        ;GET THE NEEDED BITS
2160            STA   XPMOV       ;STORE NEW NUMBER
2170            LDA   #$05        ;HIGH BYTE OF STARTING LINE MINUS 40
2180            STA   SCRNLO+1
2190            STA   SCRNHI+1
2200            LDA   #$D9        ;HIGH BYTE OF COLOR RAM
2210            STA   COLRLO+1
2220            STA   COLRHI+1
2230            LDA   #$68        ;LOW BYTE OF STARTING LINE MINUS 40
2240            STA   SCRNLO
2250            TXA
2260            BMI   RLMOVE      ;IF WRAPPED PASSED ZERO, BRANCH
2270            BPL   LRMOVE      ;DIDN'T WRAP, XPMOV WAS JUST TOO BIG
2280 BITRTS     RTS
2290 ;****** LEFT TO RIGHT MOVE ******
2300 RLMOVE     LDX   #9          ;##### NUMBER OF ROWS #####
2310 RLLP1      LDY   #$FF        ;COLUMN 0 MINUS 1
2320            LDA   SCRNLO
2330            CLC               ;GOTO NEXT ROW
2340            ADC   #40
2350            STA   SCRNLO
2360            STA   SCRNHI
2370            STA   COLRLO
2380            STA   COLRHI
2390            BCC   RLLP2       ;NO ADJUSTMENT WAS NEEDED, SO BRANCH
2400            INC   SCRNLO+1
2410            INC   SCRNHI+1
2420            INC   COLRLO+1
2430            INC   COLRHI+1
2440 RLLP2      INC   SCRNHI      ;ADJUST FO THE ONE CHARACTER OFFSET IN
MOVE
2450            INC   COLRHI
2460 RLLP3      INY               ;GOTO NEXT CHARACTER
2470            LDA   (SCRNHI),Y;GET A CHARACTER
2480            STA   (SCRNLO),Y;STORE CHARACTER IN NEW LOCATION
2490            LDA   (COLRHI),Y;GET A COLOR
2500            STA   (COLRLO),Y;STORE COLOR IN NEW LOCATION
2510            CPY   #39         ;IS ROW DONE?
2520            BNE   RLLP3       ;NO, SO BRANCH
2530            DEX
2540            BNE   RLLP1
2550            STY   COLUMN
2560            BEQ   YERPLT
2570 ;****** LEFT TO RIGHT MOVE ******
2580 LRMOVE     LDX   #9          ;##### NUMBER OF ROWS #####
2590 LRLP1      LDY   #38         ;COLUMN 38
2600            LDA   SCRNLO
2610            CLC               ;GOTO NEXT ROW
2620            ADC   #40
2630            STA   SCRNLO
2640            STA   SCRNHI
2650            STA   COLRLO
2660            STA   COLRHI
2670            BCC   LRLP2       ;NO ADJUST NEEDED, SO BRANCH
2680            INC   SCRNLO+1
```

```
2690            INC  SCRNHI+1
2700            INC  COLRLO+1
2710            INC  COLRHI+1
2720 LRLP2      INC  SCRNHI      ;ADJUST FOR THE ONE BYTE OFFSET IN MOVE
2730            INC  COLRHI
2740 LRLP3      LDA  (SCRNLO),Y;GET A CHARACTER
2750            STA  (SCRNHI),Y;PUT CHARACTER IN NEW LOCATION
2760            LDA  (COLRLO),Y;GET A COLOR
2770            STA  (COLRHI),Y;STORE COLOR IN NEW LOCATION
2780            DEY              ;NEXT ONE?
2790            BPL  LRLP3       ;NO, SO BRANCH
2800            DEX
2810            BNE  LRLP1       ;GOTO NEXT ROW IF NOT DONE
2820            STX  COLUMN
2830 ;****** WRAP CHARACTERS FROM ONE EDGE TO THE OTHER ******
2840 YERPLT     LDX  #9          ;NUMBER OF ROWS
2850            LDY  COLUMN
2860            LDA  #$05        ;HIGH BYTE OF TOP LINE MINUS 40
2870            STA  SCRNLO+1
2880            LDA  #$68        ;LOW BYTE OF TOP LINE MINUS 40
2890            STA  SCRNLO
2900 NPLP1      LDA  SCRNLO
2910            CLC              ;GO TO NEXT ROW
2920            ADC  #40
2930            STA  SCRNLO
2940            LDA  SCRNLO+1    ;AJDUST HIGH BYTE IF PAGE WAS CROSSED
2950            ADC  #0
2960            STA  SCRNLO+1
2970            TYA
2980            EOR  #39
2990            TAY
3000            LDA  (SCRNLO),Y
3010            LDY  COLUMN
3020            STA  (SCRNLO),Y
3030            TAY
3040            LDA  (COLRLO),Y
3050            LDY  COLUMN
3060            STA  (COLRLO),Y
3070            DEX
3080            BNE  NPLP1       ;DO IT AGAIN IF NOT DONE
3090            RTS
3100 ;****** RASTER INTERUPT DRIVEN ROUTINE ******
3110 RASTWT     LDA  VICIRQ      ;CLEAR THE IRQ
3120            STA  VICIRQ
3130            AND  #$01        ;IS IT A RASTER INTERUPT
3140            BEQ  RASLP1      ;NO, SO BRANCH
3150            LDA  FLAG1       ;GET THE RASTER TOGGLE FLAG
3160            EOR  #$40        ;CALC. NEW RASTER LINE
3190            STA  FLAG1       ;STORE THE NEW STATUS
3200            STA  RASTER      ;GET THE NEXT RASTER LINE THAT YOU WANT
3210                             ;AN INTERUPT ON
3220            CMP  #$C2        ;##### IS RASTER AT THE BOTTOM? #####
3230            BNE  RASLP1      ;YES, SO BRANCH
3240            LDA  XPMOV       ;GET SCROLX LOOKALIKE
3250            AND  #$07
3260            ORA  #$C0        ;SET OTHER BITS AS REQUIRED
```

```
                STA   SCROLX      ;STORE IT IN THE VIC'S X SCROLL LOC.
                STA   DELAY1      ;SET DELAY1 FLAG TO BE <> 0
                BMI   RASLP2      ;ALWAYS
     RASLP1     LDA   #$C0        ;SET SCROLX TO 0
                STA   SCROLX
     RASLP2     PLA               ;RESET THE STACK
                TAY
                PLA
                TAX
                PLA
                RTI
     ;
     ;*** SCREEN DATA   FOR EXAMPLE ONLY ***
     LINE1      .BYT     'THIS AREA DOES NOT'
     LINE2      .BYT     '      MOVE        '
     LINE3      .BYT     '   THIS AREA IS   '
     LINE4      .BYT     '    STATIONARY    '
     LINE5      .BYT     'THIS AREA SCROLLS'
     LINE6      .BYT     'WEEE..HERE WE GO!'
     .END
```

SOFTWARE APPLICATION
NOTE 3005

Authors: Joe McEnerney and Eric Cotton

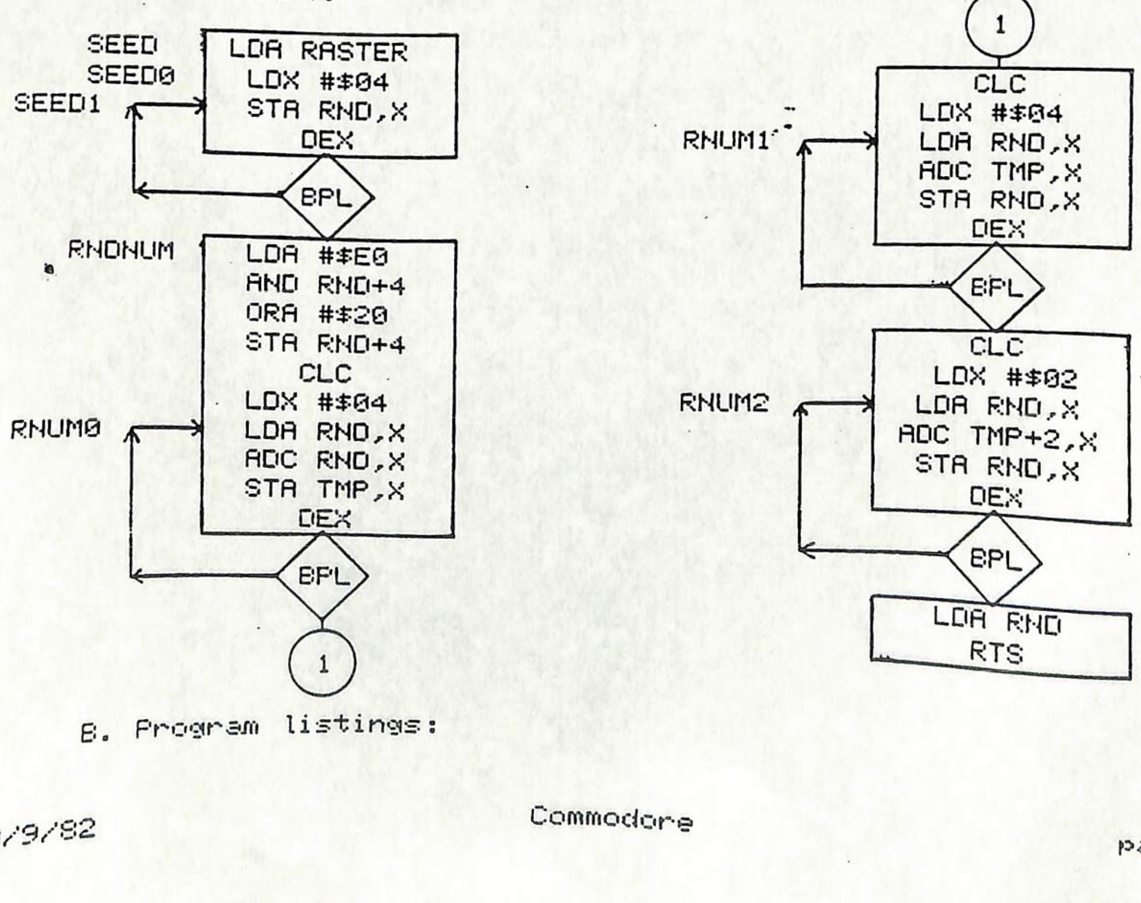
Subject: Random Number Generator

Television Standard: NTSC or PAL

I. Abstract: This subroutine generates psuedo-random numbers by the congruence x(n+1) = (3 + 2↑17) * x(n) mod(2↑35). x(0) can be any non-zero value. The routine will force this seed value to be odd to insure the maximum period. This routine operates in binary and the results will be a 35 bit binary number residing in the most significant 35 bits of a 5 byte field. The period of the sequence is 8,589,934,592 and the most significant byte is most random. A time based seed can be obtained by calling the routine called 'SEED'. The generator is normally seeded once. Subsequent calls to 'RNDNUM' will produce new random values. At RTS time the accumulator will contain the most significant byte.

II. EXPOSITION:

A. Flow Chart:



B. Program listings:

1. Source:

```
.PAGE 'RANDOM'
;
;**** RANDOM NUMBER GENERATOR ****
;
;CALL 'SEED' FOR TIME BASED SEED
;(THE GENERATOR IS NORMALLY SEEDED ONCE)
;SUBSEQUENT CALLS TO 'RNDNUM'
;WILL PRODUCE NEW RANDOM VALUES.
;
*=$0002                     ;OR ELSEWHERE ON ZERO PAGE
;VARIABLES
RND     *=*+5               ;RANDOM NUMBER
TMP     *=*+5               ;TEMPORARY STORAGE
;
;CONSTANT
RASTER =$D012               ;VIC REGISTER WITH RASTER LSB'S
;
*=$C000                     ;OR ELSEWHERE
;
SEED    LDA RASTER          ;USE RASTER AS A SEED
SEED0   LDX #$04            ;OR ANY OTHER VALUE IN REG A
SEED1   STA RND,X           ;STORE SEED VALUE IN ALL BYTES
        DEX
        BPL SEED1
RNDNUM  LDA #$E0            ;MASK OFF LSB'S AND INSURE
        AND RND+4           ;THAT UPPER 35 BITS ARE AN
        ORA #$20            ;ODD BINARY NUMBER
        STA RND+4
        CLC                 ;BE SURE DECIMAL MODE IS CLEAR
        LDX #$04            ;ADD THE ARRAY RND(+0,...,+4)
RNUM0   LDA RND,X           ;TO RND(+0,...,+4) AND STORE
        ADC RND,X           ;RESULTS IN TMP(+0,...,+4). THIS
        STA TMP,X           ;MAKES [TMP] = 2 * [RND]
        DEX                 ;THIS ACCOUNTS FOR ONE OF THE 17
        BPL RNUM0           ;REQUIRED LEFT SHIFTS NEEDED LATER.
        CLC
        LDX #$04            ;ADD RND(+0,...,+4) TO
RNUM1   LDA RND,X           ;TMP(+0,...,+4) AND STORE RESULTS
        ADC TMP,X           ;IN RND(+0,...,+4). THIS MAKES
        STA RND,X           ;[RND] = 3 * [RND]
        DEX
        BPL RNUM1
        CLC
        LDX #$02            ;LEFT SHIFT TMP(+0,...,+4) 16 ADDITIONAL
RNUM2   LDA RND,X           ;TIMES AND ADD TO RND(+0,...,+4) PUTTING
        ADC TMP+2,X         ;THE RESULTS IN RND(+0,...,+4).
        STA RND,X
        DEX
        BPL RNUM2
        LDA RND             ;LOAD THE A REG WITH THE MOST RANDOM
        RTS                 ;PART OF THE RESULTS.
;
.END
```

2. Hex dump (as assembled at $C000):

```
.: 2000 AD 12 D0 A2 04 95 02 CA 10 FB A9 E0 25 06 09 20
.: 2010 85 06 18 A2 04 B5 02 75 02 95 07 CA 10 F7 18 A2
.: 2020 04 B5 02 75 07 95 02 CA 10 F7 18 A2 02 B5 02 75
.: 2030 09 95 02 CA 10 F7 A5 02 60
```

3. Data statements (as assembled at $C000):

```
1000 data 173,18,208,162,4,149,2,202,16,251,169,224,37,6,9,32
1010 data 133,6,24,162,4,181,2,117,2,149,7,202,16,247,24,162
1020 data 4,181,2,117,7,149,2,202,16,247,24,162,2,181,2,117
1030 data 9,149,2,202,16,247,165,2,96
```

C. Memory/Register requirements: The routine requires 57 ($39) bytes of memory. Also, 10 bytes for variables are needed (preferably on zero page). The routine also requires use of the accumulator and x register.

D. Worst case execution time is 355 ($163) cycles (346.85 micro-seconds) when initially seeded and 301 ($12D) cycles (294.08 micro-seconds) each time RNDNUM is called.

E. Limitations: Because the high order byte has the longest period, it is most useful when a random byte is needed.

F. Initialization: This random number generator need only be seeded once. For a time-based seed call SEED; otherwise load the accumulator with an alternate value (0 through 255) and call SEED0.

G. Note: The random number generator on the Sound Interface Device (SID) may be used as an alternative to the routine presented in this Application Note. The SID generator is not adequate for many applications since it will fail the equidistributive test. Further, use of voice 3 may be limited. Refer to the 6581 SID chip specification for more information.

H. Example: The following example demonstrates the use of the random number generator. Its purpose is to simulate throws of a pair of dice. Wen executed, hitting the F1 function key will roll the dice and display their values on the top-middle of the screen. Pushing the F3 function key will cause a break to the monitor. (Make sure the $8000 version of VICMON -XVM4.8- is loaded.) Note that the first operation of the program is to seed the program with the 8 LSB's of the raster value.

The starting address of this example is $C000.

```
1000 .PAGE 'APP3005EX'
1010 ;
1020 ;**** EXAMPLE OF RANDOM NUMBER GENERATOR ****
1030 ;        *** SIMULATE A THROW OF DICE ***
1040 ;
1050 *=$0002
1060 ;VARIABLES
1070 RND     *=*+5           ;RANDOM NUMBER
1080 TMP     *=*+5           ;TEMPORARY STORAGE
1090 ;
1100 ;CONSTANTS
1110 RASTER =$D012           ;VIC REGISTER WITH RASTER LSB'S
1120 GETIN  =$FFE4           ;KERNAL ROUTINE: GET A CHAR. FROM KEYBOARD BUFFE
1130 ;
1140 *=$C000
1150 ;
1160 EXAM    JSR SEED        ;SEED RANDOM NUMBER GENERATOR WITH RASTER VALUE
1170 EX0     JSR GETIN       ;SEE IF A KEY WAS PRESSED
1180         CMP #133        ;WAS IT THE 'F1' KEY?
1190         BEQ EX1         ;IF SO THEN BRANCH TO 'THROW' THE DICE
1200         CMP #134        ;WAS IT THE 'F3' KEY?
1210         BNE EX0         ;IF NOT THEN CHECK KEYS AGAIN
1220         BRK             ;  ELSE BREAK
1230 ;
1240 EX1     JSR EX2         ;ROLL THE FIRST 'DIE'...
1250         STA $0411       ;...AND PUT ITS VALUE ON THE SCREEN
1260         JSR EX2         ;ROLL THE SECOND 'DIE'...
1270         STA $0413       ;...AND PUT IT ON THE SCREEN
1280         JMP EX0
1290 ;
1300 EX2     JSR RNDNUM      ;GET A RANDOM NUMBER
1310         BEQ EX2         ;IF IT IS A 0 THEN GO BACK AND TRY AGAIN
1320         CMP #$07        ;IF IT IS GREATER THAN 6 THEN
1330         BCS EX2         ;  GO BACK AND TRY AGAIN
1340         CLC             ;ADD OFFSET TO GET APPROPRIATE SCREEN VALUE
1350         ADC #$30
1360         RTS
1370 ;
1380 ;
1390 SEED    LDA RASTER      ;USE RASTER AS A SEED
1400 SEED0   LDX #4          ;OR ANY OTHER VALUE IN REG A
1410 SEED1   STA RND,X       ;STORE SEED VALUE IN ALL BYTES
1420         DEX
1430         BPL SEED1
1440 RNDNUM  LDA #$E0        ;MASK OFF LSB'S AND INSURE
1450         AND RND+4       ;THAT UPPER 35 BITS IS AN
1460         ORA #$20        ;ODD BINARY NUMBER
1470         STA RND+4
1480         CLC             ;BE SURE DECIMAL MODE IS CLEAR
1490         LDX #4          ;ADD THE ARRAY RND(+0,...,+4)
1500 RNUM0   LDA RND,X       ;TO RND(+0,...,+4) AND STORE
1510         ADC RND,X       ;RESULTS IN TMP(+0,...,+4). THIS
1520         STA TMP,X       ;MAKES [TMP] = 2 * [RND]
1530         DEX             ;THIS ACCOUNTS FOR ONE OF THE 17
1540         BPL RNUM0       ;REQUIRED LEFT SHIFTS NEEDED LATER.
1550         CLC
```

```
1560          LDX #4        ;ADD RND(+0,...,+4) TO
1570 RNUM1    LDA RND,X     ;TMP(+0,...,+4) AND STORE RESULTS
1580          ADC TMP,X     ;IN RND(+0,...,+4). THIS MAKES
1590          STA RND,X     ;[RND] = 3 * [RND]
1600          DEX
1610          BPL RNUM1
1620          CLC
1630          LDX #2        ;LEFT SHIFT TMP(+0,...,+4) 16 ADDITIONAL
1640 RNUM2    LDA RND,X     ;TIMES AND ADD TO RND(+0,...,+4) PUTTING
1650          ADC TMP+2,X   ;THE RESULTS IN RND(+0,...,+4).
1660          STA RND,X
1670          DEX
1680          BPL RNUM2
1690          LDA RND       ;LOAD THE A REG WITH THE MOST RANDOM
1700          RTS           ;PART OF THE RESULTS.
1710 ;
1720 .END
```

SOFTWARE APPLICATION NOTE 4001

Author: Bill Hindorff & Joe McEnerney

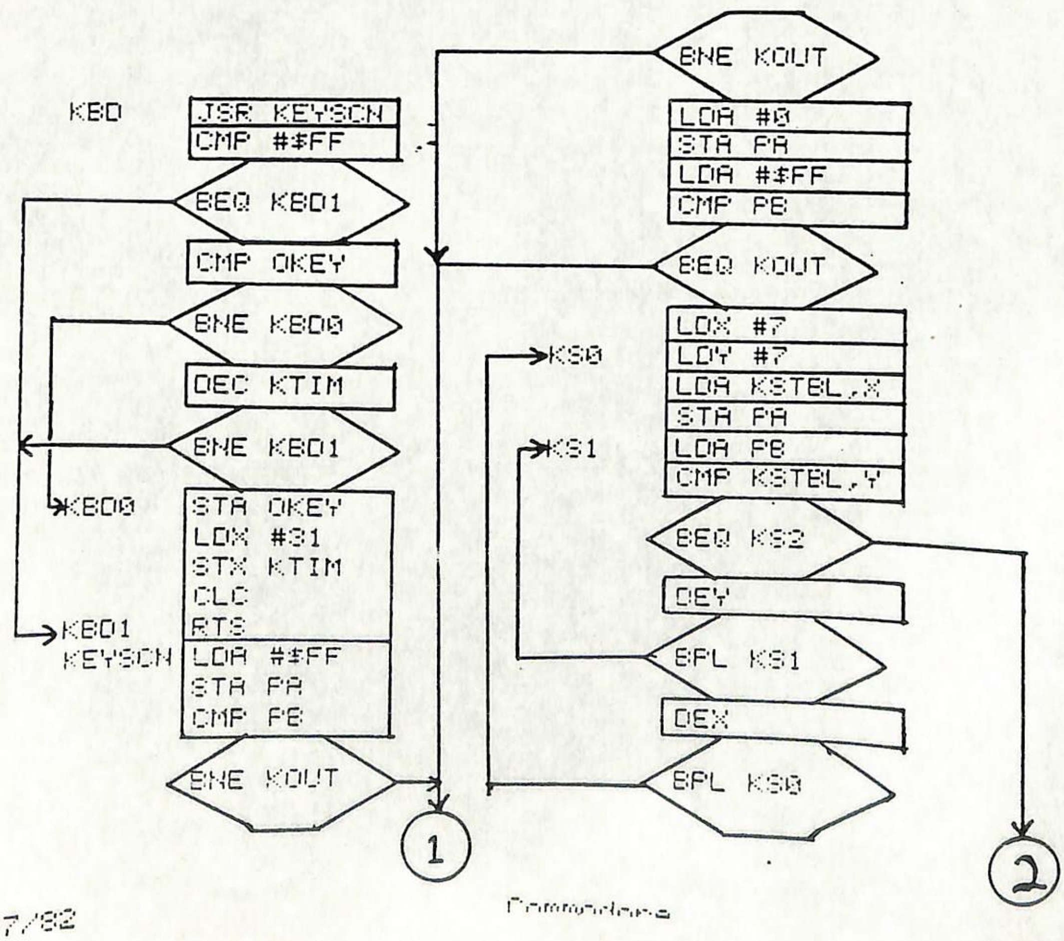
Subject: Keyboard Scanning routine

Television Standard: NTSC or PAL

Abstract: This routine reads and decodes keyboard data from ports A ($DC00) and B ($DC01) of CIA #1 (MOS 6526 chip). It is not designed to be called more than once per 1/60th of a second, and as a result, is well suited to an IRQ routine. A key on the matrix which is held down for more than half a second will auto-repeat. Upon return from the subroutine, the accumulator and a memory location defined by the variable OKEY will have been set to the row and column of the current key pressed. The upper four bits contain the row and the lower four bits contain the column. The carry is set if either no new key has been pressed, or a key has not been held down for half a second.

Exposition:

a) Flow chart



```
2 1
        TXA
KOUT    RTS
KS2     TXA
        ASL A
        ASL A
        ASL A
        ASL A
        STY TY
        ORA TY
        RTS
```

b) Diagrams: The following diagram illustrates the keyboard matrix associated with the Commodore 64 and the Max.

PORT B ($DC01)

PORT A	0	1	2	3	4	5	6	7
0	DEL	RET	CSR RGT	F7	F1	F3	F5	CSR DN
1	3	W	A	4	Z	S	E	L SFT
2	5	R	D	6	C	F	T	X
3	7	Y	G	8	B	H	U	V
4	9	I	J	0	M	K	O	N
5	+	P	L	−	.	:	@	,
6	L	*	;	HOME	R SFT	=	↑	/
7	1	←	CTRL	2	SPC	C	Q	STOP

c) Program listing as assembled at $C000

1) Source listing:

```
1000 .PAGE  KBD 7/13REV
1002 PA=$DC00
1003 PB=$DC01
1004 *=$62
1006 OKEY *=*+1
1007 KTIM *=*+1
1008 TY *=*+1
1009 *=$C200
1018 KBD     JSR KEYSCN    ;THIS ROUTINE HANDLES AUTO REPEAT
1020         CMP #$FF      ;OF KEYS HELD DOWN FOR MORE THAN
1030         BEQ KBD1      ;.5 SECONDS. IT SHOULD NOT BE CALLED
1040         CMP OKEY      ;MORE THAN ONCE PER 60TH OF A SECOND.
1050         BNE KBD0      ;A COUNTER KTIM COUNTS FRAMES
1060         DEC KTIM      ;(1/60THS OF A SECOND).  OKEY SHOULD
1070         BNE KBD1      ;BE SET TO $FF PRIOR TO FIRST USE.
1080 KBD0    STA OKEY      ;THE USER SHOULD TAKE CARE TO PREVENT
1090         LDX #31       ;MULTIPLE KEY DETECTION BY CALLING
1100         STX KTIM      ;KBD ONLY ONCE PER FRAME.  THIS WILL
1110         CLC           ;RESOLVE KEY BOUNCE PROBLEMS.
1120 KBD1    RTS           ;A VALID KEY HAS BEEN RECEIVED IF
1130 KEYSCN  LDA #$FF      ;THE CARRY IS ZERO (C=0) UPON RETURN.
1140         STA PA
1150         CMP PB
1160         BNE KOUT      ;BUTTON B OR JOYSTICK B (CODE-$FF)
1170         CMP PA
1180         BNE KOUT      ;BUTTON A OR JOYSTICK A (CODE-$FF)
1190         LDA #0
1200         STA PA
1210         LDA #$FF
1220         CMP PB
1230         BEQ KOUT      ;NO KEY DEPRESSED (CODE-$FF)
1240         LDX #7        ;MAIN SCANNING ALGORITHM
1250 KS0     LDY #7
1260         LDA KSTBL,X   ;KSTBL IS AN AUXILIARY TABLE
1270         STA PA        ;OF 8 BYTES EACH HAVING ONLY
1280 KS1     LDA PB        ;ONE ZERO BIT. EACH BYTE IS
1290         CMP KSTBL,Y   ;USED TO ENABLE A KEY MATRIX ROW
1300         BEQ KS2       ;AND LOOK FOR A KEY DOWN ON A
1310         DEY           ;COLUMN BASIS. THE COMBINATION
1320         BPL KS1       ;PRODUCES 64 ROW/COLUMN INTER-
1330         DEX           ;SECTIONS THAT CORRESPOND TO
1340         BPL KS0       ;THE PHYSICAL KEYBOARD.
1350         TXA           ;NO LEGAL KEY FOUND DURING SCAN
1360 KOUT    RTS           ;(CODE-$FF)
1370 KS2     TXA           ;THIS PART OF THE ROUTINE TAKES
1380         ASL A         ;THE ROW NUMBER (X) AND THE COLUMN
1390         ASL A         ;NUMBER (Y) AND PRODUCES A ONE BYTE
1400         ASL A         ;CODE IN THE ACCUMULATOR. THE MOST
1410         ASL A         ;SIGNIFICANT NYBBLE OF ACC
1420         STY TY        ;IS THE ROW NUMBER AND THE LEAST
1430         ORA TY        ;SIGNIFICANT NYBBLE IS THE COLUMN
1440         RTS           ;NUMBER
```

```
1450 KSTBL   .BYTE $FE,$FD,$FB,$F7,$EF,$DF,$BF,$7F
1460 .END
```

2) Hex Dump:

```
.: 2200 20 17 C2 C9 FF F0 0F C5 62 D0 04 C6 63 D0 07 85
.: 2210 62 A2 1F 86 63 18 60 A9 FF 8D 00 DC CD 01 DC D0
.: 2220 2A CD 00 DC D0 25 A9 00 8D 00 DC A9 FF CD 01 DC
.: 2230 F0 19 A2 07 A0 07 BD 56 C2 8D 00 DC AD 01 DC D9
.: 2240 56 C2 F0 08 88 10 F5 CA 10 EA 8A 60 8A 0A 0A 0A
.: 2250 0A 84 64 05 64 60 FE FD FB F7 EF DF BF 7F 00 00
```

3) Data Statements:

```
DATA 32,23,194,201,255,240,15,197,98,208,4,198,99,208
DATA 7,133,98,162,31,134,99,24,96,169,255,141,0,220,205
DATA 1,220,208,42,205,0,220,208,37,169,0,141,0,220,169
DATA 255,205,1,220,240,25,162,7,160,7,189,86,194,141,0
DATA 220,173,1,220,217,86,194,240,8,136,16,245,202,16
DATA 234,138,96,138,10,10,10,10,132,100,5,100,96,254
DATA 253,251,247,239,223,191,127
```

d) Memory/Register requirements: This routine requires 94 ($5E) bytes of memory. It uses the accumulator, X, and Y registers and 3 bytes of storage.

e) Worst case execution time is 1185 cycles (or 1161.30 μsecs on a 1.02 MHz system).

f) Limitations: A valid key has been pressed if and only if the carry is clear upon returning. Otherwise, the data in the accumulator and OKEY should be ignored. The routine can not determine if the keyboard or joystick pair caused a keypress.

g) Prior to using this subroutine, OKEY should be initialized to an $FF.

h) Example:

This example waits for a valid key to be pressed then displays the row and column of the key pressed.

```
1000 .PAGE EXAMPLE
1010 PA=$DC00
```

```
1020 CHROUT=$FFD2
1030 PB=$DC01
1040 *=$62
1050 OKEY *=*+1
1060 KTIM *=*+1
1070 TY *=*+1
1080 ;***********************************
1090 ;***********************************
1100 *=$C200
1110          SEI
1120          LDA #<KBD      ;SET UP AN IRQ VECTOR
1130          STA $0314      ;TO THE KEYBOARD EXAMPLE
1140          LDA #>KBD
1150          STA $0315
1160          LDA #$FF       ;INITIALIZE OKEY
1170          STA OKEY
1180          CLI
1200 KEYCK    LDA TY         ;IF TY IS NON-ZERO,
1210          BNE KEYCK      ;THEN THERE IS NO
1220          LDA OKEY       ;VALID KEY
1230          AND #$F0
1240          LSR A
1250          LSR A
1260          LSR A
1270          LSR A
1280          ORA #$30
1290          STA $0413      ;PUT THE ROW # TO
1300          LDA OKEY       ;THE SCREEN
1310          AND #$0F
1320          ORA #$30
1330          STA $0414      ;PUT THE COLUMN TO SCREEN
1340          JMP KEYCK      ;GO AND DO IT AGAIN
1350 ;***********************************
1360 ;***********************************
1370 .PAGE    KBD 8/27REV
1380 KBDI     LDA #$FF       ;INITIALIZE OKEY
1390          STA OKEY
1400 KBD      JSR KEYSCN     ;THIS ROUTINE HANDLES AUTO REPEAT
1410          CMP #$FF       ;OF KEYS HELD DOWN FOR MORE THAN
1420          BEQ KBD1       ;.5 SECONDS. IT SHOULD NOT BE CALLED
1430          CMP OKEY       ;MORE THAN ONCE PER 1/60TH OF A SECOND.
1440          BNE KBD0       ;A COUNTER KTIM IS USED TO COUNT FRAMES
1450          DEC KTIM       ;(1/60THS OF A SECOND).  OKEY SHOULD
1460          BNE KBD1       ;BE SET TO $FF PRIOR TO FIRST USE.
1470 KBD0     STA OKEY       ;THE USER SHOULD TAKE CARE TO PREVENT
1480          LDX #31        ;MULTIPLE KEY DETECTION BY CALLING
1490          STX KTIM       ;KBD ONLY ONCE PER FRAME.  THIS WILL
1500          CLC            ;RESOLVE KEY BOUNCE PROBLEMS.
1510 KBD1     LDA #$00
1520          ROR A
1530          STA TY         ;PUT THE CARRY IN TY
1540          LDA $DC0D      ;CLEAR THE 6526 INTERRUPTS
1550          PLA
1560          TAY
1570          PLA            ;RESTORE A, X, AND Y
```

```
1570         PLA             ;RESTORE A, X, AND Y
1580         TAX
1590         PLA
1600         RTI             ;A VALID KEY HAS BEEN RECEIVED IF
1610 KEYSCN  LDA #$FF        ;THE CARRY IS ZERO (C=0) UPON RETURN.
1620         STA PA
1630         CMP PB
1640         BNE KOUT        ;BUTTON B OR JOYSTICK B (CODE-$FF)
1650         CMP PA
1660         BNE KOUT        ;BUTTON A OR JOYSTICK A (CODE-$FF)
1670         LDA #0
1680         STA PA
1690         LDA #$FF
1700         CMP PB
1710         BEQ KOUT        ;NO KEY DEPRESSED (CODE-$FF)
1720         LDX #7          ;MAIN SCANNING ALGORITHM
1730 KS0     LDY #7
1740         LDA KSTBL,X     ;KSTBL IS AN AUXILIARY TABLE
1750         STA PA          ;OF 8 BYTES EACH HAVING ONLY
1760 KS1     LDA PB          ;ONE ZERO BIT. EACH BYTE IS
1770         CMP KSTBL,Y     ;USED TO ENABLE A KEY MATRIX ROW
1780         BEQ KS2         ;AND LOOK FOR A KEY DOWN ON A
1790         DEY             ;COLUMN BASIS. THE COMBINATION
1800         BPL KS1         ;PRODUCES 64 ROW/COLUMN INTER-
1810         DEX             ;SECTIONS THAT CORRESPOND TO
1820         BPL KS0         ;THE PHYSICAL KEYBOARD
1830         TXA             ;NO LEGAL KEY FOUND DURING SCAN
1840 KOUT    RTS             ;(CODE-$FF).
1850 KS2     TXA             ;THIS PART OF THE ROUTINE TAKES
1860         ASL A           ;THE ROW NUMBER (X) AND THE COLUMN
1870         ASL A           ;NUMBER (Y) AND PRODUCES A ONE BYTE
1880         ASL A           ;CODE IN THE ACCUMULATOR. THE MOST
1890         ASL A           ;SIGNIFICANT NYBBLE OF THE ACCUMULATOR
1900         STY TY          ;IS THE ROW NUMBER AND THE LEAST
1910         ORA TY          ;SIGNIFICANT NYBBLE IS THE COLUMN
1920         RTS             ;NUMBER
1930 KSTBL   .BYTE $FE,$FD,$FB,$F7,$EF,$DF,$BF,$7F
1940 .END
```

SOFTWARE APPLICATION NOTE 4002

Author: Bill Hindorff & Joe McEnerney

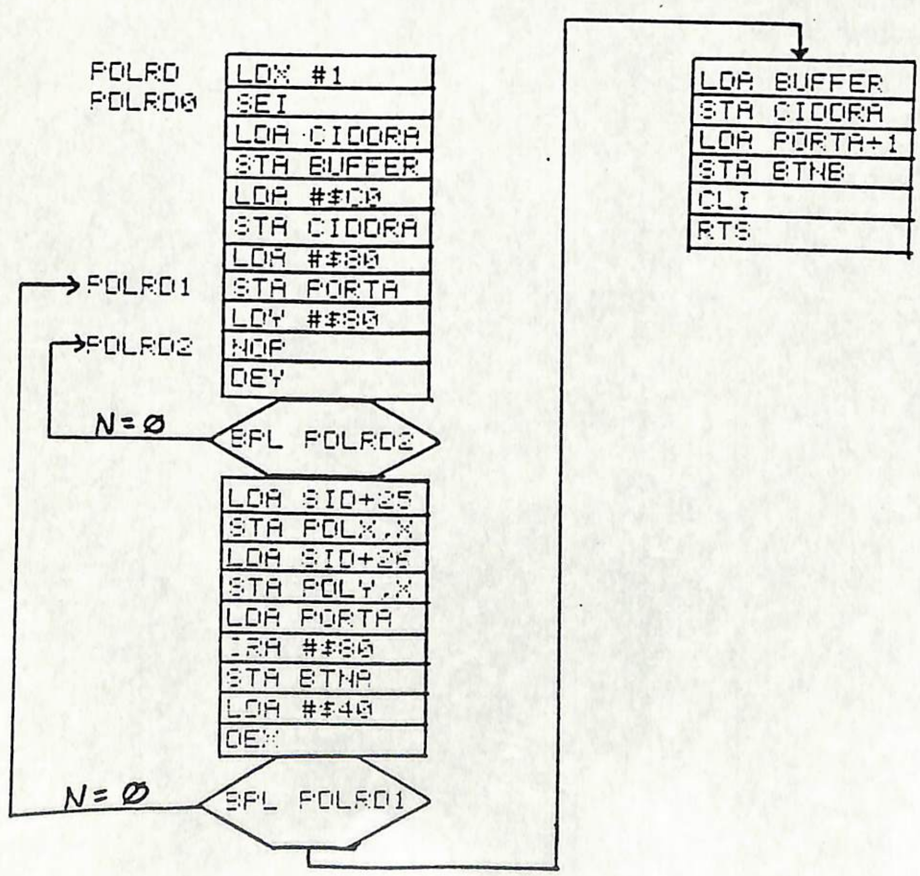
Subject: Analog Joystick/Four Paddle Read

Television Standard: NTSC or PAL

Abstract: This routine reads and decodes the paddle data from the SID (MOS 6581 chip) and reads the fire button data from ports A and B of CIA #1 (MOS 6526 chip). The standard entry point will read all four paddles (equivalent to two analog joysticks) and store the values. It also reads ports A ($DC00) and B ($DC01) and stores the values obtained. Bits two (2) and three (3) of $DC00 and $DC01 reflect the status of paddle fire buttons attached to ports A and B respectively. A second entry point, which reads paddles attached only to port A, is allowed provided the X register is conditioned to zero before the routine is called.

Exposition:

a) Flow chart



b) Diagrams: The following diagram illustrates the bit patterns associated with paddle fire buttons.

```
           Port A ($DC00)    and/or   Port B ($DC01)
      ---------------------------------------------
                  3       2        1        0
            ----------------------------------
           !Butn Y!Butn X! none  ! none  !
            ----------------------------------
```

c) Program listing as assembled at $C000

1) Assembly:

```
1000 ;*************************************************************
1010 ;* FOUR PADDLE READ ROUTINE (CAN ALSO BE USED FOR TWO)
1020 ;*************************************************************
1030 ;AUTHOR - BILL HINDORFF
1040 PORTA=$DC00
1050 CIDDRA=$DC02
1060 SID=$D400
1070 *=$C100
1080 BUFFER *=*+1
1090 POLX *=*+2
1100 POLY *=*+2
1110 BTNA *=*+1
1120 BTNB *=*+1
1130 *=$C000
1140 POLRD
1150      LDX #1           ;FOR FOUR PADDLES OR TWO ANALOG JOYSTICKS
1160 POLRD0                ;ENTRY POINT FOR ONE PAIR (CONDITION X
1170      SEI              ; TO ZERO FIRST...X=0)
1180      LDA CIDDRA       ;GET CURRENT VALUE OF DDR
1190      STA BUFFER       ;SAVE IT AWAY
1200      LDA #$C0
1210      STA CIDDRA       ;SET PORT A FOR INPUT
1220      LDA #$80
1230 POLRD1
1240      STA PORTA        ;ADDRESS A PAIR OF PADDLES
1250      LDY #$80         ;WAIT A WHILE
1260 POLRD2
1270      NOP
1280      DEY
1290      BPL POLRD2
1300      LDA SID+25       ;GET X VALUE
1310      STA POLX,X
1320      LDA SID+26       ;GET Y VALUE
1330      STA POLY,X
1340      LDA PORTA        ;TIME TO READ PADDLE FIRE BUTTONS
1350      ORA #$80         ;MAKE IT THE SAME AS OTHER PAIR
1360      STA BTNA         ;BIT 2 IS POL X, BIT 3 IS POL Y
```

```
1370     LDA #$40
1380     DEX              ;ALL PAIRS DONE?
1390     BPL POLRD1       ;NO
1400     LDA BUFFER
1410     STA CIDDRA       ;RESTORE PREVIOUS VALUE OF DDR
1420     LDA PORTA+1      ;FOR 2ND PAIR -
1430     STA BTNB         ;BIT 2 IS POL X, BIT 3 IS POL Y
1440     CLI
1450     RTS
1460 .END
```

2) Hex Dump:

```
.: C000 A2 01 78 AD 02 DC 8D 00 C1 A9 C0 8D 02 DC A9 80
.: C010 8D 00 DC A0 80 EA 88 10 FC AD 19 D4 9D 01 C1 AD
.: C020 1A D4 9D 03 C1 AD 00 DC 09 80 8D 05 C1 A9 40 CA
.: 2030 10 DE AD 00 C1 8D 02 DC AD 01 DC 8D 06 C1 58 60
```

3) Data Statements:

```
DATA 162,1,120,173,2,220,141,0,193,169,192,141,2,220,169
DATA 128,141,0,220,160,128,234,136,16,252,173,25,212,157
DATA 1,193,173,26,212,157,3,193,173,0,220,9,128,141,5,193
DATA 169,64,202,16,222,173,0,193,141,2,220,173,1,220,141
DATA 6,193,88,96
```

d) Memory/Register requirements: This routine requires 63 ($3F) bytes of memory. It uses the accumulator, X, and Y registers and 7 bytes of storage.

e) Worst case execution time is 1907 ($773) cycles (or 1868.86 µsecs on a 1.02 MHz system).

f) Limitations: This routine will not condition the two bytes which contain paddle fire button data. The button combinations produce data as follows:

BUTTONS PUSHED	HEX VALUE	DECIMAL VALUE
NONE	$FF	255
PADDLE X	$FE	251
PADDLE Y	$F7	247
BOTH	$F3	243

g) Prior to using this subroutine for a single pair of paddles, be sure you have conditioned the x register to a zero

h) Example: The following BASIC program POKES the paddle reading subroutine into memory starting at location 49152 ($C000). It then calls the subroutine and displays the paddle data on the screen.

```
 10 C=12*4096:REM SET PADDLE ROUTINE START
 11 REM POKE IN THE PADDLE READING ROUTINE
 15 FORI=0TO62:READA:POKEC+I,A:NEXT
 20 SYSC:REM CALL THE PADDLE ROUTINE
 30 P1=PEEK(C+257):REM SET PADDLE ONE VALUE
 40 P2=PEEK(C+258):REM  "      "   TWO   "
 50 P3=PEEK(C+259):REM  "      "  THREE  "
 60 P4=PEEK(C+260):REM  "      "   FOUR  "
 61 REM READ FIRE BUTTON STATUS
 62 S1=PEEK(C+261):S2=PEEK(C+262)
 70 PRINTP1,P2,P3,P4:REM PRINT PADDLE VALUES
 72 REM PRINT FIRE BUTTON VALUES
 75 PRINT:PRINT"fire a ";S1,"fire b ";S2
 80 FORW=1TO50:NEXT:REM WAIT A WHILE
 90 PRINT"S":PRINT:GOTO 20:REM CLEAR THE SCREEN AND DO AGAIN
 95 REM DATA FOR MACHINE CODE ROUTINE
 100 DATA162,1,120,173,2,220,141,0,193,169,192,141,2,220,169
 110 DATA128,141,0,220,160,128,234,136,16,252,173,25,212,157
 120 DATA1,193,173,26,212,157,3,193,78,0,220,202,16,232,173
 130 DATA0,193,141,2,220,173,0,220,141,5,193,173,1,220,141
 140 DATA6,193,88,96
READY.
```

SOFTWARE APPLICATION
NOTE 4003

Author: Bill Hindorff

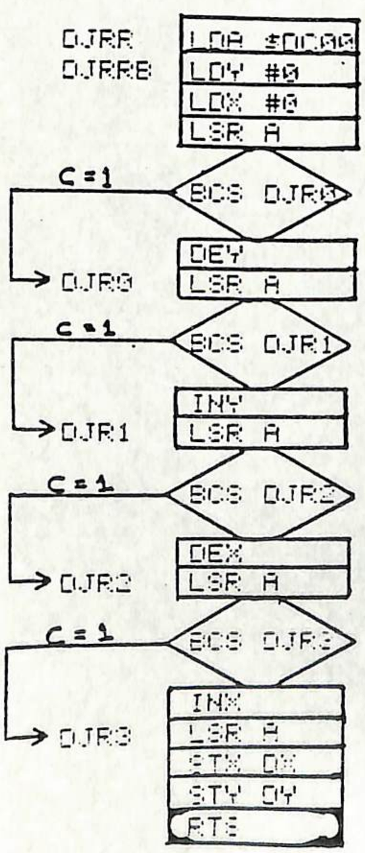
Subject: Digital Joystick Read

Television Standard: NTSC or PAL

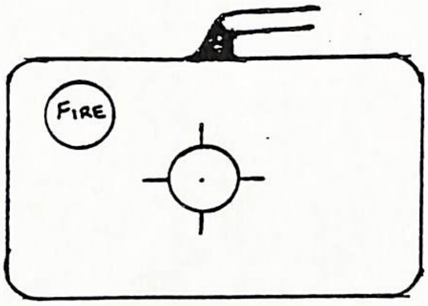
Abstract: This routine reads and decodes the joystick/firebutton data as given in the accumulator. The accumulator is assumed to be a value from port A or port B of CIA #1. The routine will set two variables, DX and DY, which represent the two's complement direction vector. I.E. $FF = -1, $00 = 0, and $01 = 1. Upon returning, the carry reflects whether or not the fire button is being pressed. Carry set (c = 1) means the button was not pushed, carry clear (c = 0) means the button has been pushed.

Exposition:

a) Flow chart



b) Diagrams: The following diagram illustrates the joystick values and which bits they represent.



(Top)

Fire Button
Bit 4

Bit 0

Bits 0 & 2 — Bits 0 & 3

Bit 2 — Bit 3

Bits 1 & 2 — Bits 1 & 3

Bit 1

c) Program listing as assembled at $C200

1) Assembly

```
1000 .PAGE (JOYSTICK.8/5)  JOYSTICK - BUTTON READ ROUTINE
1010 ;
1020 ;AUTHOR - BILL HINDORFF
1030 ;
1040 DX=$C110
1050 DY=$C111
1060 *=$C200
1070 DJRR    LDA $DC00 ;  (ENTRY POINT FOR PORT A ONLY)
1080 DJRRB   LDY #0    ;THIS ROUTINE READS AND DECODES THE
1090         LDX #0    ;JOYSTICK/FIREBUTTON INPUT DATA IN
1100         LSR A     ;THE ACCUMULATOR. THE LEAST SIGNIFICANT
1110         BCS DJR0  ;5 BITS CONTAIN THE SWITCH CLOSURE
1120         DEY       ;INFORMATION. IF A SWITCH IS CLOSED THEN IT
1130 DJR0    LSR A     ;PRODUCES A ZERO BIT. AN OPEN SWITCH
1140         BCS DJR1  ;PRODUCES A ONE BIT. THE JOYSTICK DIR-
1150         INY       ;ECTIONS ARE RIGHT, LEFT, FORWARD, BACKWARD
1160 DJR1    LSR A     ;BIT3=RIGHT, BIT2=LEFT, BIT1=BACKWARD,
1170         BCS DJR2  ;BIT0=FORWARD AND BIT4=FIRE BUTTON.
1180         DEX       ;AT RTS TIME DX AND DY CONTAIN 2'S COMPLIMENT
1190 DJR2    LSR A     ;DIRECTION NUMBERS I.E. $FF=-1, $00=0, $01=1.
1200         BCS DJR3  ;DX=-1 (MOVE RIGHT), DX=1 (MOVE LEFT),
1210         INX       ;DX=0 (NO X CHANGE). DY=-1 (MOVE UP SCREEN),
1220 DJR3    LSR A     ;DY=1 (MOVE DOWN SCREEN), DY=0 (NO Y CHANGE).
1230         STX DX    ;THE FORWARD JOYSTICK POSITION CORRESPONDS
1240         STY DY    ;TO MOVE UP THE SCREEN AND THE BACKWARD
1250         RTS       ;POSITION TO MOVE DOWN SCREEN.
1260 ;
1270 ;AT RTS TIME THE CARRY FLAG CONTAINS THE FIRE BUTTON STATE.
1280 ;IF C=1 THEN BUTTON NOT PRESSED. IF C=0 THEN PRESSED.
1290 .END
```

2) Hex Dump:

```
.: C200 A2 00 A0 00 AD 00 DC 4A B0 01 88 4A B0 01 C8 4A
.: C210 B0 01 CA 4A B0 01 E8 4A 8E 10 C1 8C 11 C1 60 8D
```

3) Data Statements:

```
DATA  162,0,160,0,173,0,220,74,176,1,136,74,176,1,200,74
DATA  176,1,202,74,176,1,232,74,142,16,193,140,17,193,96
```

d) Memory/Register requirements: The routine uses 31 ($1F) bytes of memory as well as 2 bytes of storage. It also uses the accumulator, X, and Y registers.

e) Worst case execution time is 48 ($30) cycles (or 47.04 usecs on a 1.02 MHz system).

f) Limitations: This routine assumes that the data direction registers have been properly set to read digital joysticks attached to port A and/or port B. The C64 does initialize the DDR's properly upon power-up to 255 (in $DC02) and 0 (in $DC03). If the DDR's are not properly set, then the routine will not yield accurate results.

g) Prior to using this subroutine, be sure that the accumulator holds the joystick reading and that the X and Y registers have been saved. Also see section (f) above.

h) Example: The following BASIC program pokes the joystick subroutine into memory starting at location 49664 ($C200). It then calls the subroutine and displays the joystick data. The only difference between the example's routine and the assembly listing is that the fire button state has been stored at location $C112.

```
10 C=12*4096
20 FOR I=0TO34:READA:POKEI+512+C,A:NEXT
30 SYS C+512
40 X=PEEK(C+272):Y=PEEK(C+273)
50 F=PEEK(C+274)
60 PRINT"delta x = ";X,"delta y = ";Y
70 IF F=1 THEN PRINT"fire"
80 PRINT"S":GOTO30
90 DATA 162,0,160,0,173,0,220,74,176,1,136,74,176,1,200,74
94 DATA 176,1,202,74,176,1,232,74,142,16,193,140,17,193
96 DATA 106,141,18,193,96
```

SOFTWARE APPLICATION

NOTE 4007

Authors: Joe McEnerney and Eric Cotton

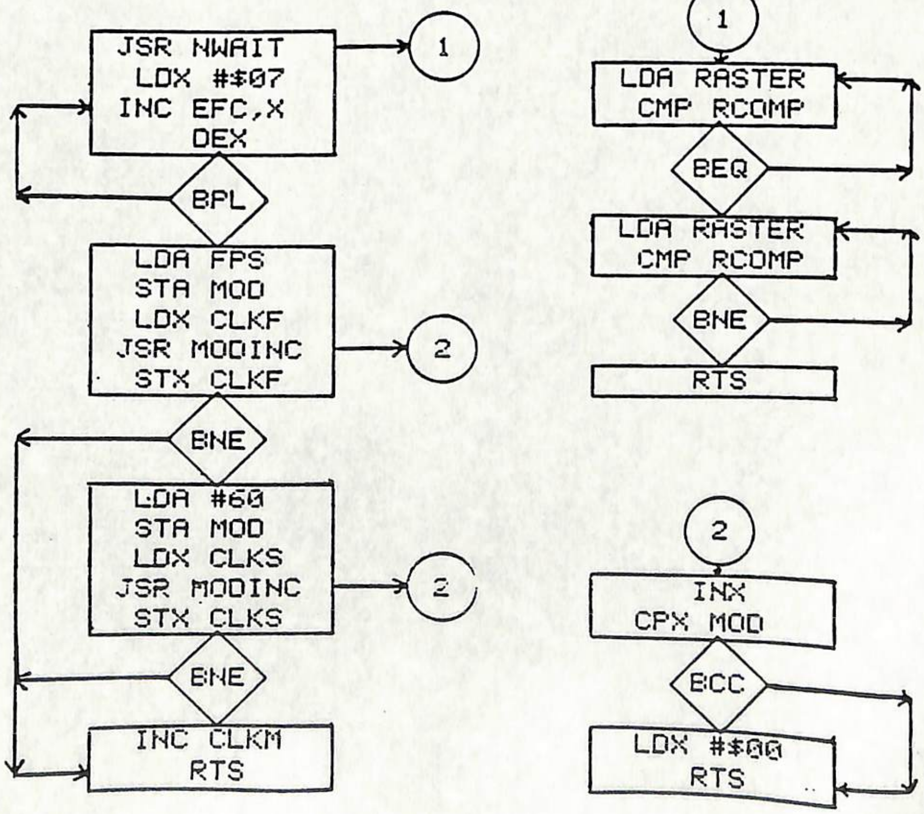
Subject: Clock Functions

Television Standard: NTSC or PAL

I. Abstract: This routine performs various clock functions, the foremost being synchronizing the logic of the calling program to the raster. In addition, it updates a set of eight frame counters, a frame clock, a seconds clock and a minutes clock. CLOCK should be called once per frame.

II. EXPOSITION:

A. Flow Chart:



B. Program listings:

1. Source:

```
1000 .PAGE 'CLOCK'
1010 ;
1020 *=$0002                    ;OR ELSEWHERE ON ZERO PAGE
1030 ;VARIABLES
1040 CLKF    *=*+1              ;FRAME CLOCK
1050 CLKM    *=*+1              ;MINUTES CLOCK
1060 CLKS    *=*+1              ;SECONDS CLOCK
1070 EFC     *=*+8              ;EVENT FRAME COUNTER
1080 FPS     *=*+1              ;FRAMES-PER-SECOND (T.V. STANDARD)
1090 MOD     *=*+1              ;MODULUS
1100 RCOMP   *=*+1              ;RASTER COMPARE VALUE
1110 ;
1120 ;CONSTANT
1130 RASTER =$D012              ;LSB'S OF RASTER VALUE
1140 ;
1150 *=$2000                    ;OR ELSEWHERE
1160 ;
1170 ;CLOCK SUBROUTINE AND SUPPORT SUBROUTINES
1180 ;
1190 CLOCK   JSR NWAIT          ;WAIT UNTIL START OF NEXT FRAME
1200         LDX #$07           ;INCREMENT THE 8 EVENT FRAME
1210 CLK0    INC EFC,X          ;COUNTERS MODULO 256
1220         DEX
1230         BPL CLK0
1240         LDA FPS            ;FRAMES PER SECOND VALUE
1250         STA MOD            ;(50-PAL, 60-NTSC)
1260         LDX CLKF
1270         JSR MODINC         ;X = X+1 MOD (FPS)
1280         STX CLKF           ;UPDATE FRAME CLOCK
1290         BNE CLK1           ;NO MODULUS CROSSING THEREFORE EXIT
1300         LDA #60            ;IF MODULUS CROSSING THEN...
1310         STA MOD            ;SET 'MOD' TO 60 FOR 60 SEC/MIN
1320         LDX CLKS
1330         JSR MODINC         ;X = X+1 MOD (60)
1340         STX CLKS           ;UPDATE SECONDS CLOCK
1350         BNE CLK1           ;NO MODULUS CROSSING THEREFORE EXIT
1360         INC CLKM           ;OTHERWISE UPDATE MINUTE CLOCK
1370 CLK1    RTS
1380 ;
1390 ;
1400 ;INC .X MODULO THE VALUE IN 'MOD'.
1410 ;
1420 MODINC INX
1430         CPX MOD            ;CHECK FOR MODULUS CROSSING
1440         BCC MODIN1         ;IF X < MOD THEN EXIT
1450         LDX #$00           ;OTHERWISE X = 0
1460 MODIN1 RTS
1470 ;
1480 ;
1490 ;WAIT UNTIL THE RASTER VALUE MISMATCHES
1500 ;THE CHOSEN COMPARE VALUE AND THEN WAIT
1510 ;UNTIL THE RASTER VALUE MATCHES THE
```

```
1520 ;COMPARE VALUE.
1530 ;
1540 NWAIT   LDA RASTER      ;CHECK RASTER
1550         CMP RCOMP       ;FOR MISMATCH
1560         BEQ NWAIT       ;IF MATCH THEN
1570 WAIT    LDA RASTER      ;CHECK RASTER AGAIN
1580         CMP RCOMP       ;FOR MATCH.
1590         BNE WAIT        ;IF MISMATCH THEN CHECK AGAIN
1600         RTS
1610 ;
1620 .END
```

2. Hex dump (as assembled at $2000):

```
.: 2000 20 2F 20 A2 07 F6 05 CA 10 FB A5 06 85 07 A6 02
.: 2010 20 27 20 86 02 D0 0F A9 3C 85 07 A6 04 20 27 20
.: 2020 86 04 D0 02 E6 03 60 E8 E4 07 90 02 A2 00 60 AD
.: 2030 12 D0 C5 08 F0 F9 AD 12 D0 C5 08 D0 F9 60
```

3. Data statements (as assembled at $2000):

```
1000 data 32,47,32,162,7,246,5,202,16,251,165,6,133,7,166,2
1010 data 32,39,32,134,2,208,15,169,60,133,7,166,4,32,39,32
1020 data 134,4,208,2,230,3,96,232,228,7,144,2,162,0,96,173
1030 data 18,208,197,8,240,249,173,18,208,197,8,208,249,96
```

C. Memory/Register requirements: The clock routine and auxiliary subroutines together occupy 62 ($3E) bytes of memory. The accumulator and the x register are used as well as 14 bytes of zero page for variables. This allows for 8 event frame counters, but more may be added if necessary (see section G below).

D. Worst case execution time is less than 16.91 milli-seconds on a NTSC system, 20.24 milli-seconds on a PAL.

E. Limitations: Because the raster compare is made to the eight least significant bits only, the user is advised to limit raster-compare values to the range 57 through 255 ($39 through $ff). This limit assumes the controlling program is to run on both PAL and NTSC television systems. If only operation on NTSC systems is intended, then the range 7 through 255 ($07 through $ff) is applicable.

F. Initialization: The clock routine requires that the frames per second value of the television be stored in the variable FPS. This number should be either 50 (for PAL) or 60 for NTSC. See lines 1390 to 1550 of the example for a technique to choose the appropriate television system that the program is running on. The variable RCOMP must be loaded with the raster compare value. Setting RCOMP equal to $FA will provide the program with the greatest amount

of time to prepare for the next frame. If the minutes/second clock is to be used then CLKF, CLKM, and CLKS should each be initialized with a zero. The event-frame-counters (EFC+0, EFC+1, EFC+2...) should be initialized in accordance with the requirements of the program. These counters are provided so that the user can time events which require many frames to display.

G. CLOCKS

1. Eight event-frame-counters are updated in this routine. CLOCK can be modified to provide more (or fewer) counters by changing the variable declaration in line 1070 of the source listing and the index amount in line 1200.

2. The 6526 Complex Interface Adapter (CIA) -the MAX Machine has one, the Commodore 64 has two- has a Time of Day clock (TOD) which may be used as an alternative to the seconds and minutes clocks provided by the CLOCK routine. See the chip specification for more information.

H. Example: The following example demonstrates one use of the clock routine. When executed, this program will display a solid sprite on the screen as well as its x and y coordinate positions. The user can then use a joystick (put in the port toward the rear of the C 64) to move the sprite to any posible x,y sprite position. The clock routine is used to update the sprite position while the raster is off screen and thus prevent the sprite from flickering.

Notes: a. To fully appreciate how synchronizing a program to the raster can improve appearance, try removing the clock routine from the example. The simplest way to do this is to remove line 1950 from the source listing.

b. Sprite movement in this example is accomplished by UNIMVX and MVSY. For an explanation of these routines refer to Application Note 1001.

c. For an explanation of the joystick routine refer to Application Note 4003.

```
1000 .PAGE 'EXAMPLE'
1010 ;
1020 ;
1030 ;      ****  EXAMPLE OF CLOCK ROUTINE  ****
1040 ; ***  VICMON SHOULD BE RESIDENT AT $8000  ***
1050 ;
1060 *=$0002
1070 ;
1080 ;VARIABLES
1090 OSH      *=*+1            ;OFFSET HIGH
1100 OSL      *=*+1            ;OFFSET LOW
1110 MODH     *=*+1            ;MODULUS HIGH
1120 MODL     *=*+1            ;MODULUS LOW
1130 TX       *=*+1            ;TEMP .X
1140 TA       *=*+1            ;TEMP .A
1150 SMSB     *=*+8            ;AUXILIARY SPRITE MSB BYTES
1160 CLKF     *=*+1            ;FRAME CLOCK
1170 CLKM     *=*+1            ;MINUTES CLOCK
```

```
1180 CLKS    *=*+1          ;SECONDS CLOCK
1190 EFC     *=*+8          ;EVENT FRAME COUNTER
1200 FPS     *=*+1          ;FRAMES-PER-SECOND (T.V. STANDARD)
1210 MOD     *=*+1          ;MODULUS
1220 RCOMP   *=*+1          ;RASTER COMPARE VALUE
1230 DX      *=*+1          ;JOYSTICK X DIRECTION OFFSET
1240 DY      *=*+1          ;JOYSTICK Y DIRECTION OFFSET
1250 ;
1260 ;CONSTANTS
1270 SP0PTR =$07F8          ;SPRITE DATA POINTER
1280 SP0X   =$D000          ;VIC REGISTER: SPRITE #0 X POSITION
1290 SP0Y   =$D001          ;VIC REGISTER: SPRITE #0 Y POSITION
1300 MSIGX  =$D010          ;VIC REGISTER: SPRITE MSB BITS
1310 RASHGH =$D011          ;VIC REGISTER: RASTER MSB
1320 RASTER =$D012          ;VIC REGISTER: RASTER LSB'S
1330 SPENA  =$D015          ;VIC REGISTER: SPRITE ENABLE
1340 XXPAND =$D01D          ;VIC REGISTER: SPRITE X-EXPANSION
1350 ;
1360         *=$C000
1370 ;
1380 EXAM    SEI            ;DISABLE IRQS
1390         LDA #50        ;ASSUME PAL UNLESS FOUND OTHERWISE
1400         STA FPS        ;  (PAL IS 50 FRAMES-PER-SEC)
1410         LDX #$01       ;SET UP PAL MODULUS MSB
1420         LDY #$F8       ;SET UP PAL MODULUS LSB
1430 EX0     LDA RASHGH     ;LOOK AT RASTER MSB IS IT A 1 ?
1440         BPL EX0        ;IF 0 THEN RASTER < 256. SO LOOK AGAIN
1450 EX1     LDA #$08       ;RASTER > 255 ...BUT...
1460         CMP RASTER     ;IS IT GREATER THAN 264 ?
1470         BCC EX2        ;IF YES THEN BRANCH. TV STD = PAL !!
1480         LDA RASHGH     ;IF NO THEN CHECK FOR MSB OF RASTER=1
1490         BMI EX1        ;IF YES THEN GOTO EX1
1500         LDX #$02       ;SET UP NTSC MODULUS MSB. TV STD = NTSC
1510         LDY #$00       ;SET UP NTSC MODULUS LSB
1520         LDA #60        ;SET UP NTSC FRAMES-PER-SECOND
1530         STA FPS        ;  (NTSC IS 60 FRAMES-PER-SECOND)
1540 EX2     STX MODH       ;STORE IN MODULUS FOR FUTURE USE
1550         STY MODL
1560 ;
1570         LDA #1         ;ENABLE AND EXPAND SPRITE 0
1580         STA SPENA
1590         STA XXPAND
1600         LDA #$7F       ;SET SPRITE Y POSITION TO 127
1610         STA SP0Y
1620         LDA #$80       ;SET SPRITE POINTER TO 128
1630         STA SP0PTR
1640         LDA #$00       ;CLEAR OUT SP0X, SMSB & MSIGX
1650         STA SP0X
1660         STA SMSB
1670         STA MSIGX
1680         LDX #62        ;SET SPRITE TO SOLID BLOCK
1690         LDA #$FF
1700 EX3     STA $2000,X    ;NOTE:- IF SPRITE POINTER=128
1710         DEX            ; THEN SPRITE ADDRESS=$2000
1720         BPL EX3
1730 ;
```

```
        JSR VTAMX     ;TRANSFER MSIGX BITS TO SMSB BYTES
        LDA #$FA      ;SET RASTER COMPARE TO JUST OFF
        STA RCOMP     ;  BOTTOM OF VISIBLE VIEWING AREA
EX4     JSR DJRR      ;READ THE JOYSTICK
        BCC EXIT      ;IF C=0 THEN FIRE BUTTON PUSHED SO EXIT
        LDA #$00      ;LOAD .A WITH SPRITE NO.
        LDX DX        ;LOAD .X WITH JOYSTICK X-DIRECTION OFFSET
        JSR UNIMVX    ;MOVE SPRITE MODULO (504-PAL OR 512-NTSC)
        JSR ATVMX     ;TRANSFER SMSB BYTES TO MSIGX BITS
        LDA #$00      ;LOAD .A WITH SPRITE NO.
        LDY DY        ;LOAD .Y WITH JOYSTICK Y-DIRECTION OFFSET
        JSR MVSY      ;MOVE THE SPRITE IN Y DIRECTION
        LDA SMSB      ;CONVERT SPRITE X COORDINATE TO ASCII
        LDY #$00      ;HEX CHARACTERS FOR DISPLAY ON TV
        JSR BTH       ;BINARY TO HEX (SCREEN ASCII) CONVERTER
        LDA SP0X      ;REPEAT FOR X LSB'S...
        LDY #$02
        JSR BTH
        LDA SP0Y      ;...AND Y
        LDY #$06
        JSR BTH
        JSR CLOCK     ;SYNCHRONIZE PROGRAM WITH RASTER
        JMP EX4
        ;
EXIT    CLI           ;CLEAR INTERUPT DISABLE...
        BRK           ;...AND BREAK TO VICMON
;
;         CONVERT BYTE TO TWO SCREEN HEX CHARACTERS
;
BTH     PHA
        AND #$0F
        JSR CONV
        STA $0411,Y
        PLA
        LSR A
        LSR A
        LSR A
        LSR A
        JSR CONV
        STA $0410,Y
        RTS
;
;         CONVERT NYBBLE TO SCREEN CHARACTER HEX
;
CONV    CMP #$0A
        BCC CONV1
        SBC #$09
        RTS
CONV1   ORA #$30
        RTS
;
;         UNIVERSAL MOVE SPRITE IN X DIRECTIONS
;
;      A REG=SPRITE NO.    X REG=OFFSET (2'S COMPLEMENT FORM)
;
UNIMVX STX TX         ;PROTECT X
```

```
2300          STX OSL        ;SET UP OFFSET LOW
2310          TAX            ;.A=X
2320          ASL A          ;.A=2*A
2330          TAY            ;.Y=A
2340          LDA #$00       ;CLEAR OFFSET HIGH
2350          STA OSH
2360          CLC
2370          LDA OSL        ;CHECK FOR NEGATIVE OFFSET
2380          BPL UMX0       ;IF POSITIVE THEN BRANCH
2390          EOR #$FF       ;PERFORM 2'S COMPLEMENT OPERATION
2400          ADC #$01
2410          STA OSL        ;PUT RESULTS IN OSL AND THEN
2420          SEC            ;FORM THE MODULAR COMPLEMENT OF
2430          LDA MODL       ;THE OFFSET BY SUBTRACTING IT
2440          SBC OSL        ;FROM THE MODULUS
2450          STA OSL
2460          LDA MODH       ;PAL 504, NTSC 512
2470          SBC OSH
2480          STA OSH
2490          CLC
2500 UMX0     LDA SP0X,Y     ;ADD OFFSET TO SPRITE X
2510          ADC OSL        ;[SMSB+X,SP0X+Y]=...
2520          STA SP0X,Y     ;...[SMSB+X,SP0X+Y]+[OSH,OSL]
2530          LDA SMSB,X
2540          ADC OSH
2550          STA SMSB,X
2560          SEC            ;IS THE SUM >= MODULUS
2570          LDA SP0X,Y     ;CHECK BY SUBTRACTING
2580          SBC MODL
2590          STA TA         ;CATCH FOR LATER USE
2600          LDA SMSB,X
2610          SBC MODH
2620          BCC UMX1       ;IF SUM < MOD THEN EXIT
2630          STA SMSB,X     ;OTHERWISE CORRECT X-COORD
2640          LDA TA
2650          STA SP0X,Y
2660 UMX1     TYA            ;RESTORE A REG
2670          LSR A
2680          LDX TX         ;RESTORE X REG
2690          RTS
2700 ;
2710 ;          TRANSFER SPRITE MSB BYTES TO MSIGX BITS
2720 ;
2730 ATVMX    LDX #$07
2740 ATV0     LDA SMSB,X
2750          LSR A
2760          ROL MSIGX
2770          DEX
2780          BPL ATV0
2790          RTS
2800 ;
2810 ;          TRANSFER SPRITE MSIGX BITS TO MSB BYTES
2820 ;
2830 VTAMX    LDX #$07
2840          LDA MSIGX
2850 VTA0     LSR SMSB,X
```

```
2860        ASL A
2870        ROL SMSB,X
2880        DEX
2890        BPL VTA0
2900        RTS
2910 ;
2920 ;         MOVE SPRITE IN Y DIRECTIONS
2930 ;
2940 MVSY   ASL A           ;AT START .A SHOULD BE LOADED WITH SPR#
2950        TAX             ;.X=2*SPR#
2960        TYA             ;OFFSET MOVED INTO .A
2970        ADC SP0Y,X      ;OFFSET IS ADDED TO SPR Y POSITION
2980        STA SP0Y,X      ;SUM IS NEW Y POSITION
2990        CMP #$1D        ;IF Y<$1D THEN C=0
3000        BCC MVSY0       ;  (AND BRANCH TO EXIT)
3010        LDA #$F9        ;  ELSE IS Y>$F9? (LAST Y ON SCREEN)
3020        CMP SP0Y,X      ;CARRY IS UPDATED ACCORDINGLY
3030 MVSY0  RTS             ;EXIT
3040 ;
3050 ;         JOYSTICK/FIRE BUTTON READ
3060 ;
3070 DJRR   LDA $DC00       ;    (GET INPUT FROM PORT A ONLY)
3080 DJRRB  LDY #0          ;READ AND DECODE THE JOYSTICK/FIRE-
3090        LDX #0          ;BUTTON INPUT DATA IN .A; THE 5 LSB'S
3100        LSR A           ;CONTAIN THE SWITCH CLOSURE INFORMATION.
3110        BCS DJR0        ;IF A SWITCH IS CLOSED THEN IT PRO-
3120        DEY             ;DUCES A 0 BIT. IF A SWITCH IS OPEN
3130 DJR0   LSR A           ;THEN IT PRODUCES A 1 BIT. THE JOY-
3140        BCS DJR1        ;STICK DIRECTIONS ARE RIGHT, LEFT,
3150        INY             ;FORWARD, BACKWARD. B3=RIGHT, B2=LEFT,
3160 DJR1   LSR A           ;B1=BACKWARD, B0=FORWARD AND B4=FIRE
3170        BCS DJR2        ;BUTTON. AT RTS TIME DX AND DY CONTAIN
3180        DEX             ;2'S COMPLEMENT DIRECTION #'S, I.E.
3190 DJR2   LSR A           ;$FF=-1, $00=0, $01=1. DX=-1 (MOVE
3200        BCS DJR3        ;LEFT), DX=1 (MOVE RIGHT), DX=0 (NO X
3210        INX             ;CHANGE). DY=-1 (MOVE UP SCREEN), DY=1
3220 DJR3   LSR A           ;(MOVE DOWN SCREEN), DY=0 (NO Y CHANGE).
3230        STX DX          ;THE FORWARD JOYSTICK POSITION CORRESPONDS
3240        STY DY          ;TO MOVE UP THE SCREEN AND THE BACKWARD
3250        RTS             ;POSITION TO MOVE DOWN SCREEN.
3260 ;
3270 ;AT RTS TIME THE CARRY FLAG CONTAINS THE FIRE BUTTON STATE.
3280 ;IF C=1 THEN BUTTON NOT PRESSED. IF C=0 THEN PRESSED.
3290 ;
3300 ;
3310 ;CLOCK SUBROUTINE AND SUPPORT SUBROUTINES
3320 ;
3330 CLOCK  JSR NWAIT       ;WAIT UNTIL START OF NEXT FRAME
3340        LDX #$07        ;INCREMENT THE 8 EVENT FRAME
3350 CLK0   INC EFC,X       ;COUNTERS MODULO 256
3360        DEX
3370        BPL CLK0
3380        LDA FPS         ;FRAMES PER SECOND VALUE
3390        STA MOD         ;(50-PAL, 60-NTSC)
3400        LDX CLKF
3410        JSR MODINC      ;X = X+1 MOD (FPS)
```

```
3420           STX CLKF         ;UPDATE FRAME CLOCK
3430           BNE CLK1         ;NO MODULUS CROSSING THEREFORE EXIT
3440           LDA #60          ;IF MODULUS CROSSING THEN..
3450           STA MOD          ;SET 'MOD' TO 60 FOR 60 SEC/MIN
3460           LDX CLKS
3470           JSR MODINC       ;X = X+1 MOD (60)
3480           STX CLKS         ;UPDATE SECONDS CLOCK
3490           BNE CLK1         ;NO MODULUS CROSSING THEREFORE EXIT
3500           INC CLKM         ;OTHERWISE UPDATE MINUTE CLOCK
3510 CLK1      RTS
3520 ;
3530 ;INC .X MODULO THE VALUE IN 'MOD'.
3540 ;
3550 MODINC INX
3560           CPX MOD          ;CHECK FOR MODULUS CROSSING
3570           BCC MODIN1       ;IF X < MOD THEN EXIT
3580           LDX #$00         ;OTHERWISE X = 0
3590 MODIN1 RTS
3600 ;
3610 ;WAIT UNTIL THE RASTER VALUE MISMATCHES
3620 ;THE CHOSEN COMPARE VALUE AND THEN WAITS
3630 ;UNTIL THE RASTER VALUE MATCHES THE
3640 ;COMPARE VALUE.
3650 ;
3660 NWAIT    LDA RASTER       ;CHECK RASTER
3670          CMP RCOMP        ;FOR MISMATCH
3680          BEQ NWAIT        ;IF MATCH THEN CHECK AGAIN
3690 WAIT     LDA RASTER       ;CHECK RASTER
3700          CMP RCOMP        ;FOR MATCH
3710          BNE WAIT         ;IF MISMATCH THEN CHECK AGAIN
3720          RTS
3730 ;
3740 .END
READY.
```

SOFTWARE APPLICATION NOTE 4000

AUTHOR : Andy Finkel

SUBJECT : Using the Commodore 64 as a Max Emulator

TV STANDARD : NTSC or PAL

ABSTRACT

The Commodore 64 can be used to emulate a Commodore Max during software development for the latter machine. This gives several advantages including the use of development tools on the 64, the ability to test the program in RAM (which ends the need to constantly write EPROMs in order to test the program), and the use of the 64 peripherals.

EXPOSITION

A brief review of the way in which MAX programs should be configured is in order:

A game program for the MAX is usually 2K bytes, 4K bytes, or 8K bytes. There is always RAM in the MAX from $0002 to $07FF for your program to use. Some programs will require an additional 2K of RAM- this memory would go from $0800 to $0FFF. The program must be assembled to run at $F800 (for the 2K size), $F000 (for the 4K size) or $E000 (for the 8K size). The program must include the 3 6510 vectors (NMI, RES, and IRQ) beginning at $FFFA for the program to operate. Graphics (both characters and sprites) must be located

either in the 4K block beginning at $F000 or the RAM from $0002 to $0FFF.

NOTE: Remember that the ROM graphics block at $F000 maps down to $3000 from the point of view of the VIC-II chip.

The object then, is to configure the Commodore 64 to match this specification. The memory configuration of the 64 is controlled by location 1. Placing a $E5 into location 1 will switch out both BASIC and the KERNAL. The MAX program must replace the KERNAL.

The other problem is to simulate the graphics-mapping effect for the VIC-II chip. This is accomplished by having the program section that goes from $F000-$FFFF also loaded at $3000-$3FFF.

There is no problem if your program requires the extra 2K of RAM, since the Commodore 64 always has RAM there.

COMPARISON OF THE MAX AND COMMODORE 64 MEMORY MAPS

The memory map of the Commodore 64 after using the MAX.EMU program is shown compared with the memory map of the Max.

```
             MAX                                       64
FFFF  -------------------                      FFFF -------------------
F800  :      ROM         :                     F800 :      RAM         :
      :     SPACE        :                          :                  :
F000  -------------------                      F000 -------------------
      :      ROM         :                          :      RAM         :
      :     SPACE        :                          :                  :
E000  -------------------                      E000 -------------------
      :    VIC CHIP      :                          :    VIC CHIP      :
      :     (I/O)        :                          :     (I/O)        :
D000  -------------------                      D000 -------------------
      :     UNUSED       :                          :      RAM         :
      :                  :                          :    (64MON)       :
A000  -------------------                      A000 -------------------
      :      ROM         :                          :      RAM         :
      :     SPACE        :                          :                  :
8000  -------------------                      8000 -------------------
      :     UNUSED       :                          :      RAM         :
      :                  :                          :                  :
4000  -------------------  xxx            xxx  4000 -------------------
      :VIC SHADOW OF     :  x              x        :      RAM         :
      : $F000--$FFFF     :  x              x        :                  :
3000  -------------------   x              x   3000 -------------------
      :     UNUSED       :  x              x        :      RAM         :
      :                  :  x              x        :(MAX.EMU prs)     :
1000  -------------------   x              x   1000 -------------------
      :   2K RAM         :  x              x        :      RAM         :
      :EXPANSION AREA    :  x              x        :                  :
0800  -------------------   x   VIC        x   0800 -------------------
      : VIDEO MATRIX     :  x   CHIP       x        : VIDEO MATRIX     :
      :& SPRITE PNTRS    :  x   MEM        x        :& SPRITE PTRS     :
0400  -------------------   x   MAP        x   0400 -------------------
      :      RAM         :  x   AREA       x        :      RAM         :
      :                  :  x              x        :                  :
0200  -------------------   x              x   0200 -------------------
      :     STACK        :  x              x        :     STACK        :
      :                  :  x              x        :                  :
0100  -------------------   x              x   0100 -------------------
      :   ZERO PAGE      :  x              x        :   ZERO PAGE      :
      :                  :  x              x        :                  :
0002  -------------------   x              x   0002 -------------------
0001  : 6510 REGISTER    :  x              x   0001 :6510 REGISTER     :
      : 6510 DDR         :  x              x        :6510 DDR          :
0000  -------------------  xxx            xxx  0000 -------------------
```

SUMMARY

1) Assemble the MAX program at $E000 for an 8K cartridge program
at $F000 for a 4K cartridge program
at $F800 for a 2K cartridge program

Remember to include the interrupt and reset vectors for the 6510 chip at $FFFA.

2) Do an OFFSET load using the HILOADER.C4 program included with the Commodore 64 Assembler Development System- use an offset of 4000. This will offset your program as follows:

MAX ADDRESS	PROGRAM WILL BE LOADED AT
$E000-$EFFF	$2000--$2FFF
$F000-$FFFF	$3000--$3FFF

3) Now LOAD the MAX.EMU program as execute it as follows:

LOAD "MAX.EMU",8,1
SYS 9*256

Your program will be transfered from $2000-$3FFF to $E000-$FFFF, the memory will be reconfigured, and execution will begin at the address specified by the RES vector in your MAX program.

Please note that the max emulator program is completely

relocatable. It can be moved anywhere, if the location occupies conflicts with another development tool. The zero page locations can be can be changed by reassembly.

SOURCE LISTING

```
1000 .PAG   'EMU.S'
1010 ;
1020 ;******************************
1030 ;* COMMODORE 64 MAX EMULATOR
1040 ;******************************
1050 ;
1060 ;VARIABLES
1070 ;
1080 *=$0002
1090 SOURCE *=*+2
1100 TARGET *=*+2
1110 ;
1120 *=$0900
1130 ;
1140 EMU    SEI                 ;DISABLE INTERRUPTS
1150        LDA #00
1160        STA SOURCE
1170        STA TARGET
1175        STA $DC0E           ;TURN OFF TIMER ON CIA #1
1180 ;
1190        LDA #$20
1200        STA SOURCE+1
1210 ;
1220        LDA #$E0
1230        STA TARGET+1
1240 ;
1250        LDA #$E5            ;TURN OFF KERNAL AND BASIC
1260        STA $01
1270 ;
1280        LDY #$00            ;NOW DO THE TRANSFER
1290 AGAIN  LDA (SOURCE),Y
1300        STA (TARGET),Y
1310        INC TARGET
1320        INC SOURCE
1330        BNE AGAIN
1340 ;
1350        INC TARGET+1
1360        INC SOURCE+1
1370 ;
1380        LDA SOURCE+1
1390        CMP #$40            ;DONE YET ?
1400        BNE AGAIN           ;NO
1410 ;
1420        JMP ($FFFC)         ;START MAX PROGRAM
1430 ;
1440 .END
```

HEX DUMP

```
.: 0900    78 A9 00 85 02 85 04 8D
.: 0908    0E DC A9 20 85 03 A9 E0
.: 0910    85 05 A9 E5 85 01 A0 00
.: 0918    B1 02 91 04 E6 04 E6 02
.: 0920    D0 F6 E6 05 E6 03 A5 03
.: 0928    C9 40 D0 EC 6C FC FF
```

DATA STATEMENTS

```
DATA   120, 169, 0, 133, 2, 133, 4, 141, 14, 220
DATA   169, 32, 133, 3, 169, 224, 133, 5, 169
DATA   229, 133, 1, 160, 0, 177, 2, 145
DATA   4, 230, 4, 230, 2, 208, 246, 230
DATA   5, 230, 3, 165, 3, 201, 64, 208
DATA   236, 108, 252, 255
```

NOTE: Remember that the emulator program is relocatable

SOFTWARE APPLICATION

NOTE 4009

Authors: Joe McEnerney and Eric Cotton

Subject: Power On Clear

Television Standard: NTSC or PAL

I. Abstract: The Power On Clear (POC) routine performs many of the setup duties necessary in most programs. Among other functions it will:

1. initialize the microprocessor stack pointer
2. initialize the 6566/67 VIC registers
3. clear the 6581 SID registers
4. determine the television standard
5. set up the I/O ports
6. set up the video and color matrices
7. stop all interrupts from the 6526 CIA
8. set up the sprite pointers
9. set up interrupt structure

All or part of this routine may be used in accordance with the requirements of the user's program. Also, the user may wish to change some of the constants to meet his particular needs.

II. EXPOSITION:

A. Flow Chart: The following chart summarizes the functions of POC and illustrates the order in which they are executed.

disable IRQ'S
clear decimal mode
set up I/O ports
set stack pointer
clear zero page
clear VIC registers
clear SID registers
set up 6510 DDR
initialize VIC registers
determine T.V. standard
clear and color screen
clear timer a control reg.
set up sprite pointers
prepare IRQ's
enable IRQ'S

8. Program listings:

1. Source

```
1000 .PAGE 'POWER ON CLEAR'
1010 ;
1020 ;****  POWER ON CLEAR ROUTINE  ****
1030 ;
1040 ;THIS ROUTINE INITIALIZES THE MICROPROCESSOR STACK POINTER,
1050 ;6566/67 VIC REGISTERS, CLEARS THE 6581 SID REGISTERS,
1060 ;DETERMINES THE TELEVISION STANDARD, SETS UP THE I/O PORTS,
1070 ;SETS UP THE VIDEO/COLOR MATRICES, STOPS ALL INTERRUPTS FROM
1080 ;THE 6526, AND SETS UP THE SPRITE POINTERS AS THE USER SPECIFIES
1090 ;
1100 *=$0002                    ;OR ELSEWHERE ON PAGE ZERO
1110 ;VARIABLES
1120 FPS    *=*+1               ;FRAMES-PER-SECOND DEPENDS ON TV STD.
1130 XSTRT  *=*+1               ;X START POSITION FOR TV STANDARD
1140 MODH   *=*+1               ;TV STD MODULUS HIGH
1150 MODL   *=*+1               ;TV STD MODULUS LOW
1160 ;
1170 ;CONSTANTS
1180 CIAIEM =$7F                ;CIA IRQ ENABLE MASK (NOTHING ENABLED)
1190 CNO    =$00                ;CHARACTER NUMBER FOR CLEAR SCREEN
1200 COLOR  =$05                ;COLOR FOR FOREGROUND NYBBLE (GREEN)
1210 SRSN   =$08                ;STARTING RAM SPRITE NUMBER
1220 VICIEM =$00                ;VIC IRQ ENABLE MASK (NOTHING ENABLED)
1230 VRTL   =$07                ;VIC REGISTER TABLE LENGTH-1
1240 VRO    =$11                ;VIC REGISTER OFFSET
1250 IRQVEC =$0314              ;C-64 IRQ VECTOR (CHANGE FOR GAME)
1260 VM     =$0400              ;VIDEO MATRIX
1270 VIC    =$D000              ;VIC II CHIP (6566/67)
1280 RASHGH =$D011              ;VIC REGISTER: RASTER MSB
1290 RASTER =$D012              ;VIC REGISTER: RASTER LSB'S
1300 SID    =$D400              ;SID I CHIP (6581)
1310 CM     =$D800              ;SCREEN COLOR NYBBLES
1320 CIA    =$DC00              ;PERIPHERAL CHIP (6526)
1330 ;
1340 *=$3000                    ;DUMMY ASSEMBLE POINT
1350 ;
1360 ;* DISABLE IRQ'S *
1370 POC    SEI
1380 ;
1390 ;* CLEAR DECIMAL MODE *
1400        CLD
1410 ;
1420 ;* SET UP I/O PORTS *
1430        LDX #$00
1440        STX CIA+3           ;SET PORT B TO INPUTS (6526)
1450        TXA                 ;.A=X
1460        DEX                 ;.X=$FF
1470        STX CIA+2           ;SET PORT A TO OUTPUTS (6526)
1480 ;
1490 ;* SET STACK POINTER *
1500        TXS                 ;.X=$FF
1510 ;
```

```
;* CLEAR PG 0 (EXCEPT 0,1), AND VIC AND SID REGISTERS *
        DEX             ;.X=$FE
POC0    STA $01,X       ;(.A=$00) CLEAR PG 0 EXCEPT 0,1
        STA VIC,X       ;CLEAR VIC REGISTERS
        STA SID,X       ;CLEAR SID REGISTERS
        DEX
        BNE POC0
;
;* SET UP 6510 DDR AT $0000 *
        LDA #47
        STA $00
;
;* INITIALIZE VIC II CHIP WITH VTBL (VIC REG TABLE) *
        LDX #VRTL       ;SET UP VIC REG TABLE LENGTH
POC1    LDA VTBL,X      ;VTBL=VIC REG TABLE
        STA VIC+VRO,X   ;VRO=VIC REGISTER OFFSET
        DEX
        BPL POC1
;
;* DETERMINE T.V. STANDARD *
        LDA #50         ;ASSUME PAL UNLESS FOUND OTHERWISE
        STA FPS         ;  (PAL IS 50 FRAMES-PER-SEC)
        LDX #$01        ;SET UP PAL MODULUS MSB
        LDY #$F8        ;SET UP PAL MODULUS LSB
POC2    LDA RASHGH      ;LOOK AT RASTER MSB: IS IT A 1?
        BPL POC2        ;IF 0 THEN RASTER < 256, SO LOOK AGAIN
POC3    LDA #$08        ;RASTER > 255 ...BUT...
        CMP RASTER      ;IS IT GREATER THAN 264?
        BCS POC4        ;IF YES THEN BRANCH. TV STD = PAL!!
        LDA RASHGH      ;IF NO THEN CHECK FOR MSB OF RASTER=1
        BMI POC3        ;IF YES THEN CHECK RASTER LSB
        INX             ;(X=2) SET UP NTSC MODULUS MSB. TV STD=NTSC
        LDY #$00        ;SET UP NTSC MODULUS LSB
        LDA #60         ;SET UP NTSC FRAMES-PER-SEC
        STA FPS         ;  (NTSC IS 60 FRAMES-PER-SEC)
POC4    STX MODHC       ;STORE IN MODULUS FOR FUTURE USE
        STY MODL
;
;* CLEAR AND COLOR THE SCREEN *
        LDX #0          ;SET UP VIDEO/COLOR MATRICES
POC5    LDA #CNO        ;CNO=CHARACTER NUMBER
        STA VM,X        ;VM=VIDEO MATRIX
        STA VM+256,X
        STA VM+512,X
        STA VM+768,X
        LDA #COLOR      ;COLOR FOR NYBBLES (GREEN)
        STA CM,X        ;CM=COLOR MATRIX (NYBBLES)
        STA CM+256,X
        STA CM+512,X
        STA CM+768,X
        DEX
        BNE POC5
;
;* CLEAR OUT THE TIMER A CONTROL REG (6526) *
;        (OPTIONAL FOR MAX)
        LDA #$00
```

```
2080          STA CIA+$E
2090 ;
2100 ;* SET UP SPRITE POINTERS *
2110          LDX #7
2120          LDY #SRSN     ;SET RAM(VM+1016+X)=SRSN+X; X=0,...,7
2130 ROC6     TYA
2140          STA VM+1016,X
2150          INY
2160          DEX
2170          BPL ROC6
2180 ;
2190 ;* SET UP IRQ STRUCTURE *
2200          LDA #VICIEM   ;LOAD VIC IRQ MASK ENABLES
2210          STA VIC+$1A   ;VIC IRQ ENABLE REG
2220          LDA #CIAIEM   ;LOAD CIA IRQ MASK ENABLES
2230          STA CIA+$D    ;CIA IRQ CONTROL REGISTER
2240          LDA CIA+$D    ;CLEAR ANY PENDING IRQS FROM 6526
2250          LDA VIC+$19   ;CLEAR ANY PENDING IRQS FROM VIC
2260          STA VIC+$19   ;VIC IRQ STATUS REGISTER
2270          LDA #<IRQDST  ;SET UP IRQ VECTOR
2280          STA IRQVEC    ;IRQVEC=$0314 FOR C-64
2290          LDA #>IRQDST
2300          STA IRQVEC+1
2310 ;
2320 ;* RE-ENABLE IRQ'S *
2330          CLI
2340 ;
2350 MAINX    JMP MAINX     ;FIX TO JUMP OVER VTEL ET CETERA
2360 IRQ      JMP (IRQVEC)  ;GAME IRQ VECTORS HERE FROM TOP OF MEMORY
2370 IRQDST   RTI           ;DUMMY IRQ ROUTINE
2380 ;
2390 ;THIS TABLE IS LOADED INTO THE VIC REGISTERS (VR0=17;
2400 ;VTRL=7) IT ENABLES DISPLAY, SETS ECM, 40 ROWS,
2410 ;Y OFFSET=1, RASTER COMP=$F9, ENABLES ALL SPRITES,
2420 ;SETS 40 COLUMNS, SETS VM=$0400, CHAR BASE=$3000
2430 ;
2440 VTEL     .BYTE $5B,$F9,$00,$00,$FF,$08,$00,$1C
2450 ;
2460     .END
```

2. Hex dump (as assembled at $3000):

```
.: 3000 78 D8 A2 00 8E 03 DC 9A CA 8E 02 DC 9A CA 95 01
.: 3010 9D 00 D0 9D 00 D4 CA D0 F5 A9 2F 85 00 A2 07 BD
.: 3020 A3 30 9D 11 D0 CA 10 F7 A9 32 85 02 A2 01 A0 F8
.: 3030 AD 11 D0 10 FB A9 08 CD 12 D0 B0 0C AD 11 D0 30
.: 3040 F4 E8 A0 00 A9 3C 85 02 86 04 84 05 A2 00 A9 00
.: 3050 9D 00 04 9D 00 05 9D 00 06 9D 00 07 A9 05 9D 00
.: 3060 D8 9D 00 D9 9D 00 DA 9D 00 DB CA D0 E1 A9 00 8D
.: 3070 0E DC A2 07 A0 08 98 9D F8 07 C8 CA 10 F8 A9 00
.: 3080 8D 1A D0 A9 7F 8D 0D DC AD 0D DC AD 19 D0 8D 19
.: 3090 D0 A9 A2 8D 14 03 A9 30 8D 15 03 58 4C 9C 30 6C
.: 30A0 14 03 40 5B F9 00 00 FF 08 00 1C
```

B. Data statements (as assembled at $3000):

```
1000 data 120,216,162,0,142,3,220,138,202,142,2,220,154,202,149,1
1010 data 157,0,208,157,0,212,202,208,245,169,47,133,0,162,2,189
1020 data 163,48,157,17,208,202,16,247,169,50,133,2,162,8,160,248
1030 data 173,17,208,16,251,169,8,205,18,208,176,12,173,17,208,48
1040 data 244,232,160,0,169,60,133,2,134,4,132,5,162,0,169,0
1050 data 157,0,4,157,0,5,157,0,6,157,0,7,169,5,157,0
1060 data 216,157,0,217,157,0,218,157,0,219,202,208,225,169,0,141
1070 data 14,220,162,7,160,8,152,157,248,7,200,202,16,248,169,0
1080 data 141,26,208,169,127,141,13,220,173,13,220,173,25,208,141,25
1090 data 208,160,162,141,20,3,169,48,141,21,3,88,76,156,48,108
2000 data 20,3,64,91,249,0,0,255,8,0,28
```

C. Memory/Register requirements: The POC routine requires 171 ($AB) bytes of memory. This includes 7 bytes for a VIC register table as well as 7 bytes for lines 2350 through 2370. In addition, 4 bytes are needed for variables, preferably located on zero page. The accumulator and the x and y registers are used.

D. Worst case execution time is less than 1/30 second on a PAL television system, 1/25 second on a PAL.

E. User Notes:

1. Lines 2070 and 2080 of the source listing are optional for programs made to run on the MAX Machine alone. CIA+$E is cleared on power-up. Such is not the case on the Commodore 64, so if CIA interrupts are to be disabled, this register must be set to 0 by the program.

2. The POC routine has been organized into sections so that the user can easily modify the routine to meet the needs of his particular program. In the source listing each section is headed by a comment (describing the function) surrounded by asterisks. If a section is not needed it can be deleted without affecting the others. However, the x register should be preserved when either "SET UP I/O PORTS" or "SET STACK POINTER" are deleted.

3. The constants and the VIC register table (VTBL) are given sample values and can be changed in accordance with the user program.

4. An indirect jump in line 2360 was provided to accommodate multiple IRQ routines.